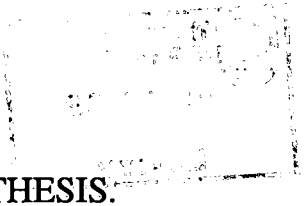
The development of an error-correcting scheme for use with a six-tone HF modem

Thesis

UNRESTRICTED

CALVIN STANFIELD BSc.

THE DEVELOPMENT OF AN ERROR-CORRECTING SCHEME FOR USE WITH A SIX-TONE HF MODEM.



PhD THESIS.

ELECTRONIC SYSTEMS ENGINEERING.

FACULTY OF TECHNOLOGY.

THE OPEN UNIVERSITY.

Date of submission: 30th September 1996
Date of award - 9th December 1997

DECEMBER 1997.

Abstract

This thesis describes the development of an error correcting system for a H.F. modem employing 6-tone Multi-Frequency Shift Keying (MFSK) as its modulation scheme. The modulation scheme was chosen to be compatible with equipment already in service and to eliminate the need to modify the existing communications infrastructure.

A convolutional code together with either Viterbi decoding or Fano decoding is chosen to provide the error correction because of the potential power of such codes and because it is possible for these combinations of code and decoding method to work with any alphabet size. To detect whether correction has been successful a Cyclic Redundancy Check (CRC) is embedded within the data block before encoding.

A method of using a convolutional code to provide variable rate is presented. The method uses a systematic code so that it is possible for the scheme to have a quick look to see if the first data transmission has been received error free. A search for good codes is undertaken and the effect the alphabet size has on the code spectra discussed. It is shown that a good generator sequence for a binary code is also a good generator sequence for non-binary codes.

To decode the convolutional code both the Viterbi maximum likelihood decoder and the Fano sequential decoder are studied. It is argued that the Fano sequential decoder is the better choice for this application because it makes better use of system resources which will be limited in the field equipment. It is also shown that the performance of multi-level codes is better than binary codes and that an alphabet size of around 6 is optimum.

The throughput of the variable rate scheme and a number of fixed rate schemes is examined. It is shown that the variable rate scheme provides the best throughput at all data rates and that the throughput improvement at the higher data rates is greatest. The effect of interleaving is also examined and results presented.

To support the variable rate scheme a protocol is developed that can be used on practical H.F. channels. The potential problems with errors on both the forward and return channel are analysed and mechanisms to deal with these built-in.

Table Of Contents

List Of Figures.

List Of Tables.

Glossary of main symbols and abreaviations used.

A_0	The number of symbols in a given input alphabet.
ACK	Acknowledgement.
A_e	The number of symbols in a given output alphabet.
AM	Amplitude Modulation.
ARQ	Automatic Repeat Request.
B_0	Normalised bandwidth.
C	A collection or group of symbols.
CDF	Column Distance Function.
C_i	The gain of the ith loop.
CRC	Cyclic Redundancy Check.
d	Distance.
d_{free}	The free distance of a convolutional code.
d_i	Column Distance Function.
DPSK	Differential Phase Shift Keying.
E	Signal energy.
f_0	The natural frequency of a resonant circuit.
FCO	Foreign & Commonwealth Office.
FEC	Forward Error Correction.
f_i	The ith frequency in a MFSK modulation scheme.
F_i	The ith forward path gain.
FSK	Frequency Shift Keying.
G	Generator Matrix.
$g^{(i)}, g^{(i)}(x)$, $g(x)$	Generator sequence vector, generator sequence vector polynomial and generator polynomial respectively.
GS	Generator Sequence.

H	Entropy - measure of average information.
HF	High Frequency.
I	Information measure.
k	The number of symbols input to the encoder during each encoding step.
L	Input sequence length.
M	The number of symbols used in the coding scheme being discussed.
m	Memory order of the convolutional code.
MFSK	Multi-Frequency Shift Keying.
MLD	Maximum Likelihood Decoding.
n	The block size.
n_A	Constraint length.
NACK	Negative acknowledgement.
PSK	Phase Shift Keying.
q	A prime.
R	Code Rate.
r, r(x)	received sequence and received polynomial respectively.
R_0	Computational Cut-Off Rate.
R_{comp}	Computational Cut-Off Rate.
T	Time.
T(x)	Generating Function.
T_0	The symbol period of a given data transmission.
t_d	Tuned filter sample period.
u, u(x)	Input sequence and polynomial respectively.
$v^{(i)}$, $v^{(i)}(x)$, v(x)	Output Vector, output vector polynomial and output polynomial

	respectively.
$\hat{v}$	The estimate of the transmitted sequence.
X^i	The ith branch gain (branch weight).

1. Thesis Summary.

The objective of the work discussed in this thesis is the development of an error correcting scheme that can be used with a new piece of communications equipment being developed. The equipment being developed is a HF modem with a modulation scheme whose characteristics are based on existing equipment. The reasons for choosing this modulation scheme are detailed in chapter 2 and are, briefly:

- The need to maintain compatibility with older equipment still in service.
- The existing modulation scheme had proved itself robust and was based on sound arguments.
- Use of the existing modulation scheme will require no change to the communications infrastructure.

The theory of finite field algebra which applies to binary and non-binary algebraic block codes shows that the symbol sets for these codes must contain q symbols where q is either a prime or a power of a prime (Lin & Costello 1983), (Michelson & Levesque 1985).

As the modulation scheme to be used in the new equipment uses 6 symbols this theory is inappropriate and a coding scheme that can cope with 6 symbols is sought. Convolutional codes combined with either Viterbi decoding

or Fano sequential decoding will work with any number base and are therefore an ideal candidate for this application.

It is important to realise that convolutional codes can be thought of as a special case of block codes where the size of the generator matrix is of infinite size. In the case where the convolutional code is truncated into blocks then there is no real distinction between convolutional codes and block codes. This latter case is considered in more detail in appendix A.

Use of the HF channel places other requirements on the coding scheme. Because the HF channel has time varying characteristics, it is advantageous for the error correcting scheme to have time varying characteristics that can match the channel. Variable rate error correcting schemes that can vary the amount of redundancy to combat channel errors provide this mechanism.

Appendices A, B & C introduce convolutional codes and two methods of decoding them. Most of the work in these appendices is text book material (Lin & Costello 1983), (Michelson & Levesque 1985), except that the schemes are described in terms of general alphabets and not just binary ones as is often the case.

Appendix A introduces convolutional codes and their properties. It also discusses encoding convolutional codes and suitable notation to describe the encoding process.

Appendix B discusses maximum likelihood decoding of convolutional codes and describes an algorithm due to Viterbi. Following this is a discussion on the performance of convolutional codes where the importance of the codes minimum free distance d_{free} is shown.

Appendix C examines an alternative *non-optimum* decoding strategy referred to as sequential decoding. Of the many sequential decoding schemes available the one concentrated on here is the Fano sequential decoder. The efficiency of sequential decoding when compared to Viterbi maximum likelihood decoding is shown using examples. Characteristics of sequential decoding are presented and the fact that the decoding time is a random variable is discussed. The Column Distance Function, which is an important distance measure for sequential decoders, is described.

Chapters 3, 4 & 5 present the variable rate scheme developed. They discuss the advantages and disadvantages of sequential decoding and maximum likelihood decoding and look at the potential throughput available from a number of schemes.

Chapter 3 discusses the advantages of using a variable rate scheme when the communication medium is the HF channel. A method of generating a variable rate code that is applicable to any convolutional code decoding method is presented and a simple protocol to use this scheme described. The protocol is a hybrid type using Forward Error Correction to attempt to correct errors introduced by the channel and a checksum to check received/decoded data for errors. The generation of the checksum is not straightforward due to the number base used and a method of working round this is given. The next section looks at the generation of a suitable generator sequence and presents the results of a number of searches for good codes. Inconsistencies are discussed and the correlation between generator sequences for different alphabets is presented.

Chapter 4 compares sequential decoding with maximum likelihood decoding. It is shown that the outright decoding performance of the maximum likelihood decoder is superior to that of the sequential decoder. However, the sequential decoder is shown to have an advantage in terms of decoding speed and decoding resources. Also to the sequential decoder's advantage is the fact that the decoding resources it requires (and to a large extent decoding speed) is independent of code memory. This advantage is exploited and it is shown that the

sequential decoders performance can be improved to the same level as the 'equivalent' maximum likelihood decoder operating on a code with memory m. The use of the term equivalent refers to the resources used and not the code used. It is shown that the sequential decoder's code memory can be increased from m to $m + \delta m$ without significantly changing the decoding time or decoding resources used.

Chapter 5 examines the throughput available from the variable rate scheme and it is seen that the throughput is better than any fixed rate scheme. The throughput is analysed for two channels:

- A simple channel with random errors ranging from 0% channel error rate through to 18% channel error rate.
- Simulated HF channels using error maps to determine the location of errors.

The error statistics of the HF Channel are examined and it is shown that many blocks are error free. It is this fact that makes the scheme developed so efficient as the first 'decoding attempt' simply examines the block and its check sum to see if it is error free. It is also shown that by increasing the rate of transmission the throughput is also increased. This is not as obvious as it may seem because it was anticipated that the increase

in error rate at the higher transmission rates would result in a net reduction in throughput. This was not true for the cases considered and it is seen that for a large percentage of the time the H.F. channel is capable of supporting low error-rate communications at data rates well above those required for this application.

Chapter 6 describes the development of a full protocol that may be used with the variable rate scheme developed. The protocol is designed to cope with errors in both the forward and return channel. It is also designed to be flexible allowing a number of unacknowledged blocks to be outstanding without reducing the system throughput. Appendix D gives full details of how this protocol was implemented.

Chapter 7 is "Conclusions And Areas Of Further Work". It picks out the salient points and results and presents them in a concise manner. It also looks at some outstanding questions and recommends areas for further work.

Chapter 8 lists all the references and acknowledges the support and help from others.

2. Introduction And History.

2.1 Motivation.

The motivation for this work comes from a need to develop a suitable error-correcting scheme for a new piece of equipment that must support existing formats. This situation arises frequently in industry and inevitably results in compromises being made. In the following sections we discuss the main characteristics of the communications equipment being used by the Foreign and Commonwealth Office (FCO) which is to be replaced. For a more detailed discussion the reader is referred to the book by Ralphs (1985) which gives excellent coverage of the development of the equipment together with general details of HF communications.

2.2 The Equipment To Be Replaced.

Since the early 1960s the FCO has used Multi-Frequency Shift Keying (MFSK) High Frequency (HF) modems as its main line communications equipment for communicating with its embassies abroad. J.D. Ralphs, who had developed the use of MFSK for the FCO, undertook a study of the modulation scheme used in the existing units (known as Piccolo units) and as a result, between 1978 and 1981, the Piccolo Mark 6 was developed.

MFSK was never adopted by the communications industry and the reasons for this are many and varied. For a start, due to the narrowband filters used by the system, a very

accurate frequency reference was required which at the time only existed inside the laboratory and was very expensive. Another influencing factor was that the development of early HF communications equipment was dominated by the intercontinental trunk telegraphy networks operating ITU-recommended systems. In this arena the trend was to use equipment that operated in the narrowest of bands and so new equipment not following this trend tended to be overlooked (although the modulation scheme used by Piccolo used narrowband filters, the overall bandwidth compared to binary DPSK schemes was rather large). Another obvious point is that binary equipment is much easier to produce, is generally less bulky and consequently less expensive.

However, MFSK was used for other projects outside the FCO and one of these designs was for a maritime application sending traffic from ship to shore. It was not believed that systems could be developed that could out-perform the trained ear of a skilled operator and yet the MFSK equipment demonstrated that this was clearly the case particularly when the signal was weak and disturbed by noise (Ralphs 1985).

2.3 Initial Design Considerations.

It became necessary to start the development of Piccolo successor equipment around 1988 as many of the components used in the existing equipment were becoming obsolete. The new system is microprocessor based and employs many

digital signal processing techniques, thus relieving the system designers of many of the constraints imposed by older systems. Some of the improved features include automatic frequency selection, automatic speed changing, automatic link establishment and a much improved man-machine-interface. Although all these features will be immediately available to those stations supplied with the new equipment it will still be necessary to communicate with stations employing the old equipment that have no automatic functions and utilise fixed modulation schemes and formats.

2.4 Backwards Compatibility Issues - General.

The main characteristics of Piccolo equipment that are relevant to this thesis are:-

1) The equipment uses 6-level MFSK as its modulation scheme and uses the HF channel as its communication medium.
2) The equipment uses narrowband filters which are realised by using orthogonal signalling where the relationship between the element duration and frequency spacing is inverse.
3) The receive and transmit equipment use synchronous communications. Combining this with the fact that narrowband filters are used generates the requirement for there to be very accurate frequency references at each end.

2.5 Backwards Compatibility Issues - Specific.

2.5.1 MFSK Over The HF Channel.

Of the modulation schemes available, only frequency modulation or phase modulation need be considered when designing HF communications equipment due to the gross inefficiencies and poor performance of amplitude modulation. When transmitting discrete data, frequency modulation reduces to the transmission of one of a discrete number of possible frequencies and is referred to as Frequency Shift Keying (FSK). Phase modulation reduces to the transmission of a single frequency whose phase is shifted by a discrete amount and is referred to as Phase Shift Keying (PSK). PSK modulation is only possible over phase-stable channels (which the HF channel is not) and therefore if PSK is to be used at all it must be differential PSK (DPSK) where the phase of the present element is measured with reference to the phase of the previous element and not with reference to some absolute phase.

In communications theory, information, I, is defined as (Shannon 1948):

$$I = \log_2 M$$

where M is the size of the source alphabet. As M increases the information conveyed in each choice rises sharply at first but as M increases beyond about 10 the rise becomes less significant. This is due to the logarithm in the definition.

If more information can be conveyed with each symbol transmitted then, for a given data rate, the duration of each element can be extended giving rise to improved demodulator performance.

For DPSK the phase shift becomes increasingly small as M increases and its suitability over a channel that is often rapidly fading and subject to multi-path effects is limited.

For an FSK scheme, as the number of choices increases so does the bandwidth. To limit the impact of this the bandwidth of the filters used by the MFSK detector must be as narrow as possible allowing the tones to be more closely grouped. Also, the choice of M should be such as to optimise the throughput for a given bandwidth. The choice of 6 levels will be explained after the following description of orthogonal signalling.

2.5.2 Orthogonal Signalling And Narrow Bandwidth.

When the original Piccolo equipment was developed, filtering technology was limited to tuned LCR (Inductor, Capacitor & Resistor) circuits.

Consider a simple parallel tuned circuit tuned to say 500Hz and its response to an applied sinusoidal signal. If a signal at 500Hz is applied and the circuit is initially discharged then the output will grow as energy is pumped into the circuit in time with its natural oscillations. If on the other hand a signal that is

slightly off-tune is applied, the output will grow at first as the applied signal seems to be in synchronism with the natural oscillations of the tuned circuit. As the synchronisation slips and the incoming and natural frequencies end up in anti-phase the output will start to fall, eventually reaching zero, at which point the output will start to grow again. The duration between these successive zeros is determined by the frequency difference between the applied signal and the tuned frequency and is given by:

$$t_d = \frac{1}{f_i - f_0}$$

where t_d is the duration in seconds, f_i is the frequency of the input signal and f_0 is the natural frequency of the tuned circuit.

This result can be used to great advantage as follows. If the M frequencies used in the modulation scheme are all separated by Δf and the output of all filters is sampled every $1/\Delta f$ seconds, it will be found that the outputs of all filters, with the exception of the one tuned to the incoming signal, will be at zero. As an example consider a system transmitting one of 6 possible frequencies f_1, f_2, f_3, f_4, f_5 & f_6 separated by 10Hz (e.g. 500Hz 510Hz 520Hz ...) and a set of LCR filters tuned to those frequencies. If a frequency at f_3 were applied to all filters their outputs would be of the form shown in figure 2-1.

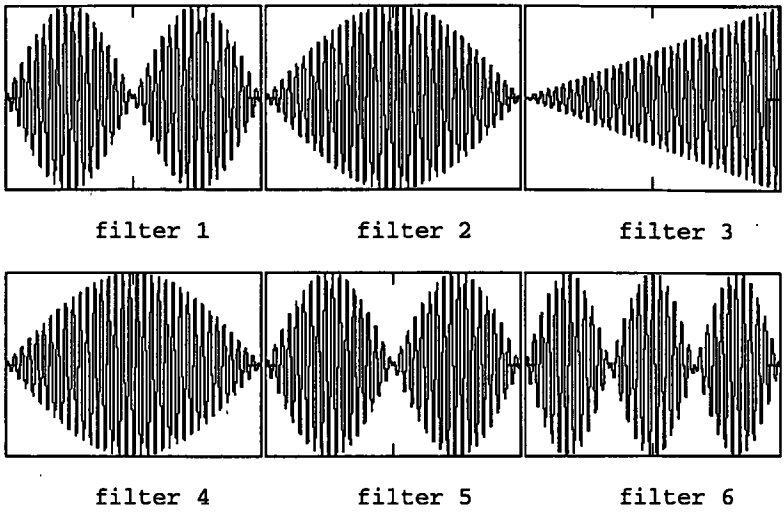


Figure 2-1 **Output of filters with f_3 applied.**

The main advantage of this system is that it allows the frequencies to be more closely grouped than would be possible with conventional filters. Obviously the closer the frequencies are grouped the longer the sample period and therefore the lower the data rate. However, by varying the number of tones used and the spacing of these tones it is possible to optimise the modulation scheme in terms of bandwidth used, data throughput and signal robustness.

2.5.3 6 Tones In The MFSK Modulation Scheme.

A suitable definition for the necessary bandwidth required to transmit a signal is that 95% of the signal energy should be conveyed. If we consider a signal consisting of a sinewave modulated "on" and "off" for alternating elements of duration *T* it is found that 95%

of the signal power is included if the second pair of sidebands is included. Therefore, if we extend the bandwidth of the MFSK scheme to include the frequencies that lie two tone spacings beyond the extreme tones we shall include 95% of the signal energy. Each tone spacing occupies $1/T$ Hz and if there are two tone spacings either side of the extreme tones then, for a system employing M tones, the bandwidth required will be $\frac{M-1+4}{T}$ Hz. The data rate of a given system is fixed by the sample period T and is given by $\frac{\log_2 M}{T}$. From this it is possible to define the normalised bandwidth, B_0, as:

$$B_0 = \frac{\left(\frac{M-1+4}{T}\right)}{\left(\frac{\log_2 M}{T}\right)} = \frac{M+3}{\log_2 M} \quad \text{Hz/bit/sec}$$

Figure 2-2 shows B_0 for various values of M and it can be seen that the value of B_0 is a minimum at around 3.5 when M is between 4 to 7 (Ralphs 1985).

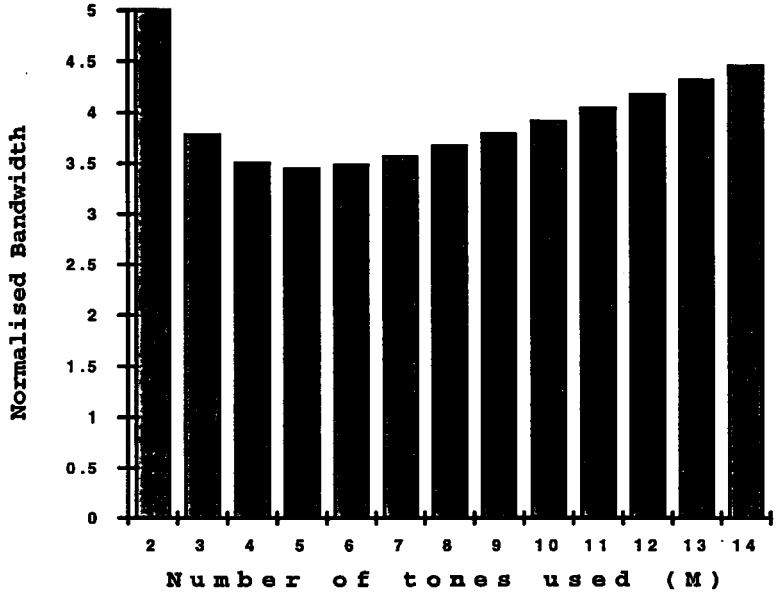


Figure 2-2 **Normalised bandwidth - vs - tones used.**

Another factor to consider is the size of the input alphabet A_0. Let each symbol from the alphabet A_0 be conveyed by C tone elements from the modulation scheme employing M frequencies. The total number of different symbols which can be generated by this arrangement is $A_e = M^C$, where A_e is the size of the output alphabet. For a given data rate the symbol period will be fixed at T_0, and since C elements must be transmitted in this period the element period is given by $T_e = T_0/C$. The required data rate is then given by:

H_0 = (bits/symbol in input alphabet)/(symbol period)

$$H_o = \frac{\log_2(A_0)}{T_0} \text{bits/sec}$$

and the actual data rate is given by:

H_e = (bits/symbol in output alphabet)/(symbol period)

$$H_e = C\left(\frac{\log_2(M)}{T_0}\right)\text{bits/sec}$$

$$= \frac{\log_2(M)}{T_e}\text{bits/sec}$$

Obviously $H_e \geq H_0$ and if this is not an equality the code includes some redundancy resulting in a waste of signal energy E, where:

$$E = \frac{H_e - H_o}{H_0} \cdot 100\%$$

The energy wasted, E, given as a percentage of the total signal energy, is shown in Table 2-1 for various values of C and M for an input alphabet A_0 of 32 characters.

C	M	M^C	E%
1	32	32	0.0
2	7	49	12.3
2	6	36	3.4
3	5	125	39.3

***Table 2-1* Modulation scheme energy loss.**

These choices represent the sensible choices for the given alphabet. Using 32 tones (as used in the early Piccolo units) gives rise to the most efficient use of power. However, this also gives rise to wide bandwidths and so is not too efficient in terms of bandwidth economy. The other choices; 5, 6 or 7 tones all give

rise to some wasted energy but the choice of 6 tones with two elements per symbol minimises this and, if the remaining 4 unmapped tone pairs are used as special control symbols for synchronising, message termination etc. then this waste is further reduced. Although the choice of 7 tones does not increase the inefficiency too much it does increase the complexity because there now have to be 7 tone filters and sampling circuits within the equipment. It is for these reasons that the 6-tone modulation scheme was chosen.

2.5.4 Synchronous/Asynchronous Communications.

There are two basic methods by which two remote units can maintain a communications link. These are either:

1. Synchronous, where all symbols are transmitted and received within a timing window defined by some central timer or
2. Asynchronous, where the sending end informs the receiving end about the transmission of a new symbol by introducing it with a start pulse.

Synchronous communications relies on there being accurate synchronism between the two ends for the complete duration of the message being transmitted. Each character is received and sampled within a precise window and the timing of that window must be maintained at each end.

Asynchronous communications is widely used throughout the communications industry and probably the most well known system is the RS232-C signalling protocol used in computer communications. This system has the advantage that the two ends can operate totally independently and can communicate with each other when necessary by announcing the transmission of each new symbol with a start pulse. The timing requirements are such that the receive clock need only be within a few percent of the transmit clock because the two ends are re-synchronised with each start pulse. It may also be necessary to use a stop pulse to allow the receive circuitry to settle before another symbol can be transmitted. The time taken to transmit these start and stop pulses could be used to transfer information instead giving rise to increased throughput. A common format is 1 start bit, 8 data bits and 1 stop bit in which 20% of the signal energy is used for the start and stop bits.

On the face of it, it would seem that the obvious choice for an HF link is asynchronous communications due to the unstable nature of the HF channel. However, the need for start and stop pulses gives rise to a considerable waste in signal energy when high transmission powers are required (or equivantly reduction in throughput) and for this reason synchronous communications are used.

As far as timing is concerned the biggest problem is the short-term instability of the path due to the constant

movement of the ionospheric layer. This movement gives rise to a variation in path length and therefore a variation in propagation delay. The effect of short-term variation in path delay can be overcome by introducing guard bands at the transition instants of the receive window. This results in the signals no longer being orthogonal but the degradation is slight and the net result is much better than the alternative reduction in throughput.

2.6 Backwards Compatibility Conclusions.

It is not absolutely necessary to maintain backwards compatibility in all situations. For example, when communicating with the older equipment (which will be in use for several years yet) the new equipment could simply select "Piccolo Mode" and emulate an old Piccolo Modem. For communication with newer equipment a more appropriate set of equipment characteristics could be used and the modem switched to another mode. However, the introduction of another mode would complicate further the design of the new modem and may even result in the need to update the whole communications infrastructure. In any case, the characteristics of the older equipment are perfectly satisfactory and the few problems they present may be overcome.

2.7 Requirements Of The New Coding Scheme.

To develop an error correcting system for the new equipment it will be necessary to choose a coding scheme that will integrate with the hardware developed. The theory of Galois Fields (Lin & Costello 1983), (Michelson & Levesque 1985) shows that algebraic coding/decoding schemes require the symbol sets to consist of q symbols where q is either a prime or a power of a prime. An example of the use of this theory is the class of cyclic block codes with BCH or RS decoding (Lin & Costello 1983), (Michelson & Levesque 1985).

Because the hardware of the new communications equipment is designed around a 6-tone modulation scheme then it is necessary to choose a coding scheme compatible with this. One option is to use a binary coding scheme and to map binary words onto groups of hexary symbols (e.g. 5 bits = 2 hexary symbols). As discussed above this results in power inefficiencies for all sensible choices of mappings. Another disadvantage of this option is that if one symbol is incorrectly decoded then a whole word, which consists of more than one symbol, is received incorrectly. This *error propagation* gets worse as the number of symbols assigned to each binary word is increased. A better solution is to develop a scheme that will work with an alphabet size of 6. Convolutionl codes together with either Viterbi decoding or Fano decoding

satisfy this requirement. It is these codes that shall be used here.

2.8 An Overview Of The Development Of Specific Areas Of Communications/Coding Theory Relevant To This Thesis.

The following paragraphs outline the development of communication theory and information theory that are directly related to the work carried out in this thesis.

Shannon's pioneering paper "A mathematical theory of communication" (1948) showed that virtually error free communication is possible by suitable encoding provided that the rate of transmission does not exceed the channel capacity. Elias (1955) considered the practical implications of Shannon's paper and sought ways of encoding/decoding this information without the enormous complexity/memory requirement implied by the size of the codespace. In doing so he introduced the concept of convolutional codes which are central to this thesis. The decoding of convolutional codes using Maximum Likelihood Decoders (MLD) calls for a large amount of computation and memory for all but the simplest (and least powerful) of codes. Wozencraft and Reiffen (1961) addressed this problem and introduced the notion of sequential decoding by considering the elimination of large parts of the decoding "tree" that were unlikely to be associated with the transmitted sequence. Fano (1963) examined sequential decoding further and developed an

algorithm that was able to decode convolutional codes simply and efficiently.

Gallager (1965) presented a simple derivation of the coding theorem (Shannon's Theorem) and derived a bound on coding error for general codes. This bound was made tight by maximising the so-called "reliability function".

One of the problems of sequential decoding is that the average time taken to decode a block of data can exceed the rate at which blocks are being received. Considering this Savage (1966) showed that the distribution of decoding time is a Pareto Distribution and that decoding can continue satisfactorily provided that the data rate does not exceed the "computational cut-off rate", which can be calculated.

Zigangirov (1966) and later Jelinek (1969) presented an alternative sequential decoding procedure called the stack or ZJ algorithm that is extremely simple.

Viterbi (1967) took Gallager's bound and applied it to convolutional codes by proposing a block code with equivalent parameters. This produced a lower bound on convolutional decoding error. He also derived an upper bound by using random coding arguments. He examined these bounds in the limiting cases and showed that convolutional codes and block codes are approximately

equal here. However, it was shown that at intermediate rates convolutional codes are better, the actual performance of any specific convolutional code being a strong function of its constraint length (or equivalently is memory order). In the same publication he presented what he then called 'An Asymptotically Optimum Decoding Algorithm' now known as the Viterbi Algorithm which is a Maximum Likelihood Decoder (MLD).

The performance of convolutional codes is determined by both their constraint length and their rate. Mandelbaum (1974) showed that the rate of a convolutional code and therefore its redundancy could be varied by deleting some of the symbols in the code (this deletion process is called puncturing). This variation allows the code's redundancy to be incremented in small steps until the level required to sustain reliable communication is reached.

Chevillat and Costello (1976) examined the relationship between a code's so-called Column Distance Function (CDF) and decoding time and concluded that the faster the growth in CDF the shorter, on average, the decoding time.

Extending the basic ideas introduced by Chevillat and Costello; Cain, Clark and Geist (1979) introduced high rate codes of rate (n-1)/n based on basic 1/n rate codes by puncturing the output. The structure of these codes

also suggest the possible use of simpler decoding schemes. Lugand and Costello (1982) examined the performance of these higher rate codes using a model of a non-stationary channel and showed that these codes have a higher throughput efficiency. A similar analysis is carried out by Wang and Lin (1983) using a code with full error correcting capabilities and the throughput efficiency is analysed for various size. A variation using soft decision decoding is also analysed by Yasuda, Kashiki and Hirate (1984).

Calderbank, Mazo and Wei (1985) presented two simple expressions that can be used to obtain a bound on the distance properties and therefore error performance of convolutional codes. Exactly which expression should be used depends on the ratio k/n where k is the number of information symbols and n is the number of symbols in the codeword. For more details on these parameters and the structure of general convolutional codes refer to appendix A.

An alternative and simple way to vary the rate of a code is to repeat some of the information already sent and to combine it in some structured way at the receiver to allow a previously undecodeable block to be decoded correctly. Chase (1985) introduced such a scheme which proved to be quite effective. Each of the blocks can be given weighting factors according to their reliability

and the time separation between successive blocks provides a type of interleaving which aids the scheme.

In the encoding and decoding of convolutional codes it is usual to return the encoder to the all zero state by making the final m symbols (where m is the memory of the encoder) in a block of data equal to zero. This "Zero-Tail" represents a waste (a reduction in rate). Ma and Wolf (1986) presented a method of eliminating this tail using "tail biting codes". Their method used the code's structure to optimise the scheme and was shown to be better than any previous scheme. A better procedure for decoding these was later presented by Wang and Bhargava (1989). Their procedure is a near MLD algorithm and in good SNRs is asymptotically equivalent to the Viterbi algorithm with one starting state and one ending state.

Codes with variable redundancy obviously lend themselves to variable rate schemes. One of the first examples of such a scheme was presented by Goodman and Farrell (1975) where a particular cyclic code is chosen to suit the prevailing channel conditions. Here the transmitter is informed of the code to use by the receiver via a feedback channel. Krishna and Morgera (1987) also developed a variable redundancy scheme based on 'partitionable matricies'. The K-M codes (as they are called) are based on block codes and rely on there being invertable matrices that can be combined to yield other

invertable matrices of larger order. These other matrices are the encoding/decoding matrices of lower rate codes and therefore provided a means of generating variable rate block codes. This scheme was later extended to include soft decision decoding by Morgera and Oduol (1989).

The first example of a variable rate scheme for convolutional codes was presented by Hagenauer (1988). He called the special class of codes 'Rate Compatible Punctured Convolutional' (RCPC) codes. The rate-compatibility came from the incremental changes made to the puncturing matrix which resulted in the structure of the decoder remaining the same for the whole scheme.

The idea of code combining introduced by Chase was further examined in connection with sequential decoding by Kallel (1988) who discovered that it had the effect of increasing the computational cut-off rate. This had the effect of making previously undecodable codes (due to their excessive average decoding time) decodable.

Another example of the now popular variable rate schemes was presented by Du, Kasahara and Namekawa (1988) using BCH codes. One of the aims of this work was to keep the block lengths the same to aid the transmission and retransmission strategy.

More recent developments in communication and information theory have included schemes that modify the decoding procedure or manipulate the code trellis in some way to simplify the decoding operation or reduce the memory requirements (Fettweis & Meyr 1989), (Anderson 1989), (Wen, Wen & Wang 1990), (Ping, Yan & Feng 1991), (Collins & Hizlan 1993). There have also been a number of variable rate schemes developed that achieve either better throughput and error correcting performance or are aimed specifically at modifying the Pareto exponent to make sequential decoding a practical solution in otherwise impractical circumstances (Fantacci 1990), (Kallel 1990), (Kallel & Haccoun 1990), (Lee 1991), (Kallel 1992), (Lee 1994).

Other recent works include the examination of decoding procedures so that certain key parameters can be optimised (Leonard & Rodger 1989), (Martins & Alves 1990), the use of Channel State Information (CSI) in the decoding process (Vucetic 1991), (Hegde, Naraghi-Pour & Chen 1994), the development of Burst Error Correcting Schemes (Schlegel & Herro 1990) and the development of Multi-level schemes employing commercially available hardware decoders (Wang *et al* 1993).

2.9 Overview Of Remaining Chapters.

Chapter 3 develops the decoding scheme around a variable rate Fano sequential decoder and explains the reasons for choosing variable rate sequential decoding.

Chapter 4 looks at the advantages and disadvantages of sequential decoding & maximum likelihood decoding and shows how sequential decoding can be made 'equivalent' to maximum likelihood decoding in terms of error-correcting power/decoding-time.

Chapter 5 analyses throughput and presents simulation data using 'real channels' showing the effectiveness of the scheme developed.

Chapter 6 looks at a practical implementation of the scheme and develops strategies to cope with errors in both the forward and return channel.

Chapter 7 draws together the salient points of the thesis and presents a concise overview of the preceding chapters. Here recommendations for further work are also given.

Chapter 8 lists the references and acknowledges the help given by others.

Appendices A, B & C introduce convolutional codes and decoding methods. Appendix A details the generation and structural properties of convolutional codes. Appendix B

describes Maximum Likelihood Decoding and Appendix C describes Sequential Decoding. As this thesis is concerned primarily with convolutional codes these appendices go into considerable detail.

Appendix D gives a detailed breakdown of the scheme developed in Chapter 6.

3. A Variable Rate Sequential Decoder

3.1 Introduction.

Chapter 2 presented an overview of the new equipment to be developed which itself is based on existing equipment still in the field. This chapter and the remainder of this thesis describe the development of an error correcting scheme suitable for the new equipment. An error correcting scheme is considered necessary for the new equipment as it uses the HF channel as its communications medium which can introduce many errors. There are many different approaches to the design of a suitable error correcting scheme each with its own advantages and disadvantages. The approach to be used here is the development of a scheme that has the following key characteristics:

- The scheme is based on the fundamental symbol set used within the equipment without the need for translation.

- The scheme varies the amount of redundancy it uses in sympathy with the prevailing channel conditions in an effort to maximise throughput.

The author already had some knowledge in the use of convolutional codes and their potential advantage over block codes to combat errors in communication systems. For this reason their use was considered for this

application. Appendix A gives an introduction to convolutional codes while appendix B and appendix C describe maximum likelihood decoding and sequential decoding of convolutional codes respectively.

This chapter describes the development of a variable rate decoder based on convolutional codes.

3.2 Software Development.

To enable the work described in the following sections of this chapter to be completed it was necessary to write a number of software applications. The applications were needed to allow experimentation and evaluation to be carried out.

Section 3.3 describes a scheme to provide variable rate encoding/decoding using a convolutional code. Before developing this scheme fully it was necessary to find good generator sequences that could be used with this scheme and software to find these generator sequences was required.

Armed with a number of suitable generator sequences, the next task was to evaluated the performance of the codes. To perform this task a Viterbi decoder application was written.

3.3 The Need For Variable Rate.

As discussed above, this thesis is concerned with the development of a coding scheme suitable for use over the HF channel. The HF channel has characteristics that vary with time, varying from almost perfect transmissions during some periods to complete channel loss during other periods. This *fading* of the channel is due to the movement of the ionospheric layer which results in constructive interference of multi-path EM-waves during the perfect periods and complete destructive interference during periods of channel loss. Because of this characteristic a coding scheme that is able to use *adaptive variable rate* is likely to perform better than either a fixed rate Forward Error Correction (FEC) scheme or an Automatic Repeat reQuest (ARQ) scheme. To see this consider a typical period on the HF channel. During the first interval the channel exhibits excellent transmission characteristics with a typical error rate of less than 1:10,000 - during this period a scheme using only a CRC block check within the transmitted frame would maximise throughput as the majority of frames would be error free. During the following period, where either the channel begins to fade or there is random interference present, the channel exhibits an error rate between 1:50 and 1:100 (representing 1 to 2 errors in a block of 100 symbols) - during this period the best error-correcting scheme would be one that uses a little

redundancy to help correct the few errors. As the channel deteriorates further (or as the interference grows) an error-correcting scheme using even more redundancy to combat errors would be needed. Eventually the channel deteriorates to such a degree that only *noise* is received and no amount of redundancy would be sufficient to correct the errors. During this latter stage the coding scheme might as well revert to a simple CRC check until the channel is seen to improve sufficiently to support further communication.

The task here is to develop such a scheme based on a convolutional code that will provide the variable rate, provide a CRC to ensure data integrity and minimise the amount of traffic that actually passes across the HF communications channel (because of the narrow bandwidth of HF circuits).

3.4 Development Of The Coding Scheme.

In the literature there are many examples of methods of introducing variable redundancy together with schemes that use variable redundancy to match the code rate to the prevailing channel conditions: (Mandelbaum 1974), (Cain, Clark & Geist 1979), (Lugand & Costello 1982), (Wang & Lin 1983), (Yasuda, Kashiki & Hirata 1984), (Chase 1985), (Krishna & Morgera 1987), (Hagenauer 1988), (Kallel 1988), (Du, Kasahara & Namekawa 1988), (Morgera &

Oduol 1989), (Fantacci 1990), (Kallel 1990), (Kallel & Haccoun 1990), (Kallel & Haccoun 1991), (Vucetic 1991), (Lee 1991), (Kallel 1992) and(Lee 1994). The reader is referred to chapter 2 section 2.7 where these are briefly discussed. Having chosen a convolutional coding scheme we must now develop an encoding/decoding method to allow variable rate. The easiest way to do this is to have a selection of good codes, one for each rate that will be used, and to select the appropriate code according to the present channel conditions. The problem with this is that on changing from say a 1/2 rate to 1/3 rate code (because the channel conditions have deteriorated) the information sent during the 1/2 rate phase must be thrown away and the new 1/3 rate code sent in its place. One solution to this problem is to produce a 1/3 rate code that is a subset of the 1/2 rate code which itself can be reduced to a one-to-one code. Figure 3-1 shows an encoder that will produce such a code.

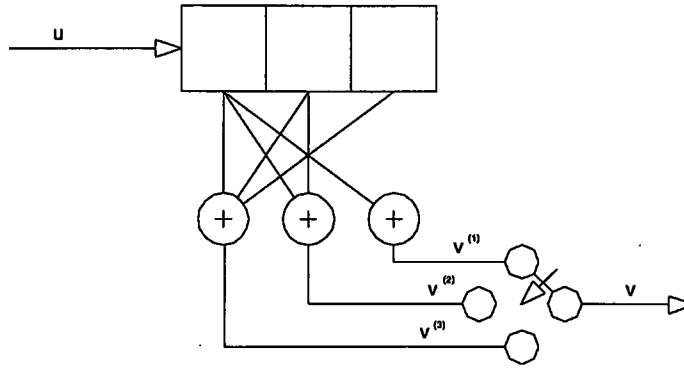


***Figure 3-1* A simple multi-rate encoder.**

Examination of figure 3-1 reveals how a convolutional encoder may produce a variable rate code. Firstly note that if output $v^{(1)}$ is taken alone then it represents the input sequence **u** with no delay and may be transmitted to the receiving site with no overhead apart from the embedded CRC symbols that will be used to see if the block is received error free. Next if outputs $v^{(1)}$ and $v^{(2)}$ are taken and output $v^{(3)}$ is ignored the corresponding sequence, when interleaved, represents a half rate convolutional code that may be transmitted to the receive site where it can be decoded using the corresponding half rate decoder. Finally, if all outputs, $v^{(1)}v^{(2)}$ and $v^{(3)}$ are transmitted to the receive site then the resulting interleaved sequence may be decoded using the appropriate third rate decoder.

An efficient way to use the encoder shown in figure 3-1 is as follows:

1) Take the next block of user data and from it calculate a CRC check. Append the CRC check to the end of the user data to make up the data block u.

2) Input the data block u to the encoder and store the output sequences $v^{(1)}$, $v^{(2)}$ and $v^{(3)}$ in separate buffers.

3) Send $v^{(1)}$ to the receive site. At the receive site recompute the CRC based on the user data portion to determine if errors occurred during transmission. If the recomputed CRC agrees with the received CRC send a

positive ACKnowledge (ACK) to the transmit site to say that the data was correctly received. If the recomputed CRC disagrees with the received CRC send a Negative ACKnowledge (NACK) to the transmit site to indicate transmission errors occurred.

4) At the transmit site, if an ACK is received repeat from step 1); otherwise send $v^{(2)}$ to the receive site. At the receive site, if a NACK was sent previously then interleave the previously received sequence ($v^{(1)}$) with the incoming sequence ($v^{(2)}$) and decode using a half rate decoder. After decoding, recompute the CRC based on the user data portion of the decoded sequence and compare this with the decoded CRC portion. If the two CRCs agree send a ACK to the transmit site otherwise send a NACK.

5) At the transmit site, if an ACK is received repeat from step 1); otherwise send $v^{(3)}$ to the receive site. At the receive site, if a NACK was sent previously then interleave the previously received sequences ($v^{(1)}$ & $v^{(2)}$) with the incoming sequence ($v^{(3)}$) and decode using a third-rate decoder. After decoding recompute the CRC based on the user data portion of the decoded sequence and compare this with the decoded CRC portion. If the two CRCs agree send a ACK to the transmit site otherwise send a NACK.

6) At the transmit site, if an ACK is received repeat from step 1; otherwise repeat from step 3. At the

receive site, if a NACK was sent previously repeat from step 1.

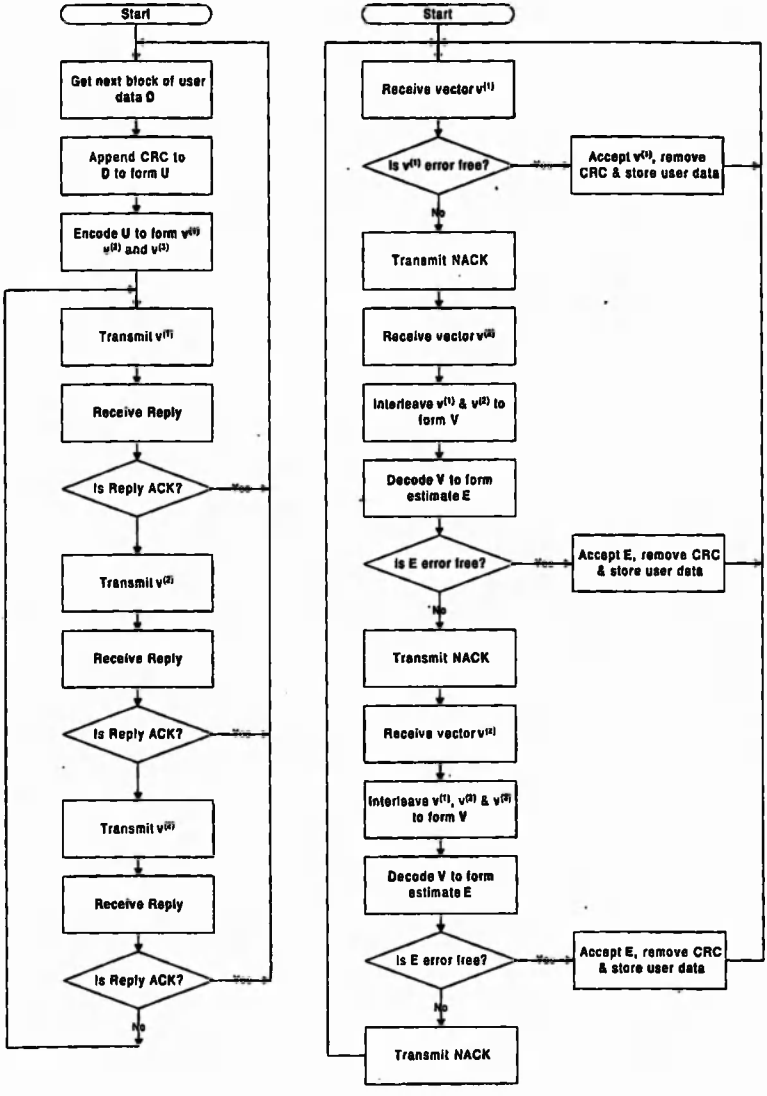
A number of comments are necessary regarding steps 1) to 6) above:

1) The transmit and receive site must keep track of of the current encoding/decoding parameters to ensure that they both remain in synchronism.

2) In step 6), if a NACK is received, the process repeats from step 3). An alternative to this would be for the transmit site to continue to cycle through its vectors ($v^{(1)}, v^{(2)}, v^{(3)}, v^{(1)}, v^{(2)}, v^{(3)}, \ldots$) with the receive site overwriting its previous version of $v^{(n)}$ and attempting a third-rate decode. The reason for adopting the former method is two-fold. Firstly it simplifies the algorithm, reducing the chance of loss of synchronism between the transmit and receive site. Secondly, if the third-rate decode failed it is likely that the channel is very poor with the previous versions of $v^{(1)}$, $v^{(2)}$ and $v^{(3)}$ containing many errors. If this is the case then it is just as likely that a fresh transmission of $v^{(1)}$ will correctly decode as a good vector $v^{(n)}$ with poor $v^{(n-1)}$ and $v^{(n-2)}$.

3) An efficient algorithm would also use a sliding window where acknowledgements for a number of previously transmitted blocks may be outstanding. This allows both the encoder and decoder to work continuously without each

having to wait for the other to transmit the next information. The process is shown concisely in the flow charts below.



(a) Transmit site (b) Receive site

Figure 3-2 **Flow charts for variable rate system.**

3.4.1 Calculation Of The CRC.

To optimise the ability of the code to detect errors it is necessary to use an efficient CRC. In a Galois Field, *GF(q)*, if an irreducible polynomial is used as a divider for another larger order polynomial the spread of remainders will exhaust all possible combinations before repeating (Michelson & Levesque 1985). This has the effect of minimising the number of sequences that will produce the same remainder. As a consequence, if the CRC is computed using such an irreducible polynomial then the likelihood of a data block with a number of transmission errors producing the same CRC will be minimum. However, because we are working in base-6 and not a prime base the theory applicable to the above argument does not hold and we must work round this problem. Fortunately this is easily overcome:

1) Calculate a 6-symbol CRC based on GF(7). The value of this CRC can range from 000000_7 to 666666_7 (0_{10} to 46655_{10}).

2) Convert the base-7 CRC value into a base-6 value. To represent this range of numbers in base 6 requires 7 symbols ($5555555_6 = 78124_{10}$).

3) Transmit the CRC portion of the data using the base-6 format.

4) At the receive site convert the base-6 CRC portion into a 6-symbol base-7 CRC.

5) Recompute the base-7 CRC from the data portion of the received transmission.

6) Compare the two CRC (recomputed and converted) to determine if errors have taken place.

3.5 Finding An Effective Variable Rate Code.

Figure 3-1 illustrates one of the basic requirements of a variable rate convolutional encoder. If the output vector $v^{(l)}$ is to be identical to the input sequence u then the generator sequence describing the code must be of the form 100xxxxxx (x = don't care) for an encoder with memory m = 2 and rate $1/n$ $(n=3)$. The consequence of this is that there are a limited number $\left(M^{(m+1)(n-1)}\right)$ of generator sequences that satisfy this requirement.

3.5.1 Finding A Good Generator Sequence.

The most important parameter for a convolutional code, in terms of its error correcting power, is its minimum free distance, d_{free}. It is shown in Appendix A, section A.1.3, that the distance spectrum of any code can be obtained from the generating function $T(x)$ and in Appendix B, section B.1.4, this is used to obtain an upper bound on the code's performance. If the code is to be decoded using a sequential decoder then another important parameter is the column distance function CDF. The importance of d_{free} is discussed in Appendix A section A.1.5.1 and Appendix B section B.1.4. The importance of

the CDF is covered in Appendix A section A.1.5.2 and Appendix C section C.1.3.2. Hence, to find a good convolutional code for our needs we must find a generator sequence that exhibits a large d_{free}, and if sequential decoding is to be used then a rapidly growing CDF would be an advantage.

To find such codes there are a number of algorithms that may be used. Here we shall use the Fast Algorithm for Searching Trees (FAST) (Cedervall & Johannesson 1989) with two modifications as detailed below:

- The algorithm is modified to deal with any size alphabet.
- The algorithm is implemented using 'forward codes' and not the 'reversed codes' suggested.

A flow diagram of the modified algorithm is shown in figure 3-3.

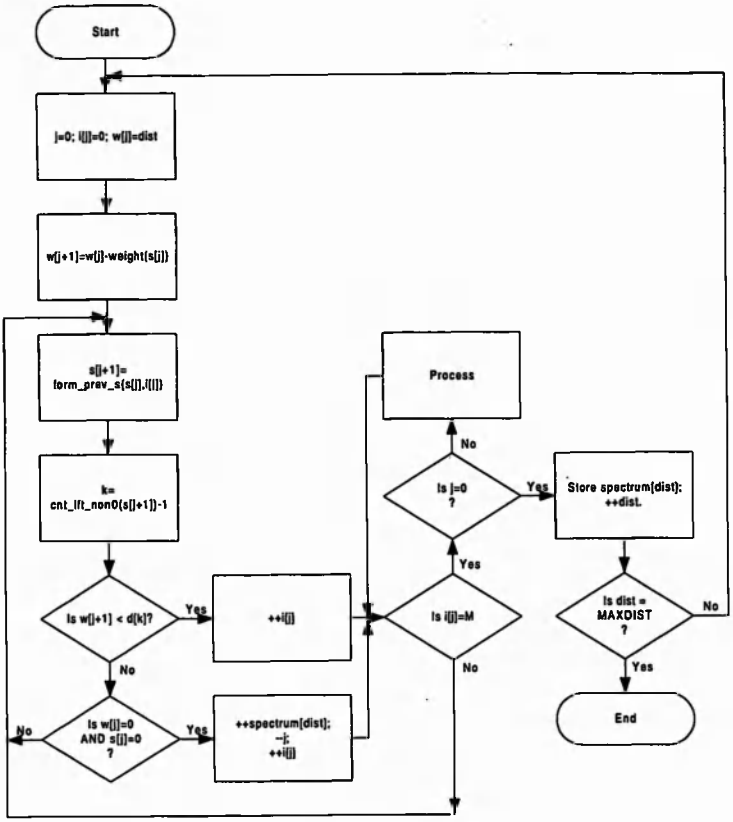


Figure 3-3 **Modified FAST algorithm.**

Functions:

weight(s) determines the weight of the codeword generated when '***s***' was entered.

form_prev_s(s,i) determines the state prior to '***s***' by knowing the input '***i***' that caused the state '***s***' to be entered.

cnt_lft_non0(s) determines the minimum number of branches that would have to be traversed from the state '***s***' to return to the all-zero root state. It does this by determining the number and position of non-zero symbols in the state-word.

Variables:

dist determines which paths are currently being searched for.

M is the number base.

j is an index used to move forward and backward in the trellis.

s[] is an array used to keep a record of the states visited.

i[] is an array used to record the input sequences associated with each branch.

w[] is an array used to keep a record of remaining weights at each state.

d[] is an array used to store the distance profile.

The algorithm in figure 3-3 was embedded within a loop that stepped though all valid generator sequences for a code with memory m=2 and rate 1/3. Invalid generator sequences were:

- Any generator sequence of the form 000xxxxxx, xxx000xxx or xxxxxx000 as these represented codes that did not use the full capabilities of the encoder.

- Any generator sequence that had an effective memory size of less than 2. To test for this condition a mask, 001001001, was used so that if the logical AND of the generator sequence and the mask gave the result 00000000 then the effective memory size was less than 2.

- Any generator sequence that generated a catastrophic code. A catastrophic code is identified by a zero weight loop in the state diagram. Note: This test was not necessary for the systematic codes used in this application as systematic codes are always non-catastrophic. However, this test was used during initial software development to ensure that the software was generating correct results. The results were compared with published results (Johannesson 1975), (Jonannesson 1977) & (Jonannesson & Paaske 1978) which include distance profiles for non-systematic codes.

The complete algorithm was run once for the code alphabets M = 2, 3, 4, 5 & 6 and the results saved. The saved data was sorted to order the distance spectra of each generator sequence so that the best generator sequence could be identified. The rules used to rank the distance spectra were as follows:

1. Codes with the largest d_{free} were considered to be the best.

2. If two codes had the same d_{free} then the code with the fewer paths of weight d_{free} was considered the better.

This is because the more paths there are of weight d_{free} the more ways there are that the most probable error patterns can occur.

3. If two codes had the same d_{free} with the same number of paths then the next spectral component was examined to see which of these was the better of the two.

4. This process continued until a difference was found. If there was no difference then the codes were declared equivalent.

Figure 3-4 shows the performance of a number of systematic codes ranked using the above rules.

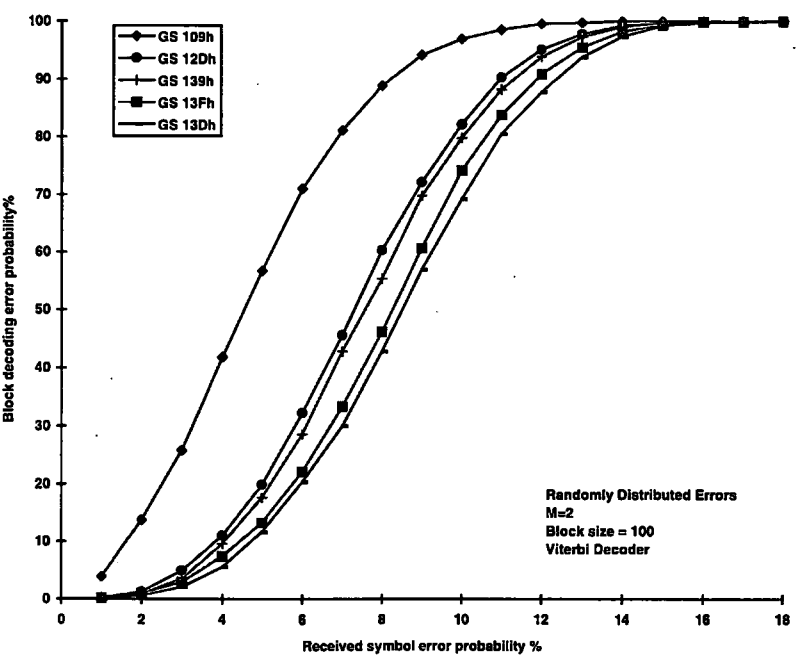


***Figure 3-4* Performance of selected generator sequences.**

The x-axis is labeled 'Received symbol error probability' and covers errors in the range 1% to 18%. It is necessary to examine the performance of the codes at error rates as high as this because the HF cahnnel is, at times, extremely poor. The properties of the HF channel are covered in more detail in chapter 8.

The generator sequences (GS) used in each trial are represented using hexidecimal notation. As an example the GS given by 100 001 001 can be re-expressed by grouping into sequences of upto 4 binary digits as 1 0000 1001 which in hexidecimal is 109_h. The generator sequences used were:

GS 109_h = 100 001 001
GS $12D_h$ = 100 101 101
GS 139_h = 100 111 001
GS $13F_h$ = 100 111 111
GS $13D_h$ = 100 111 101

Weight	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GS=														
109_h	0	0	1	0	0	3	0	0	9	0	0	27	0	0
$12D_h$	0	0	0	0	1	1	1	1	1	3	6	10	15	21
139_h	0	0	0	0	1	1	0	1	3	2	3	7	7	10
$13F_h$	0	0	0	0	0	1	1	1	3	2	5	8	10	19
$13D_h$	0	0	0	0	0	1	0	2	0	4	0	9	0	20

***Table 3-1* Distance spectra for selected generator sequences.**

It can be seen from figure 3-4 and table 3-1 that, for the selected codes shown, the codes with the 'better' spectra do indeed perform better. However, these codes have been *selected* to illustrate this fact and there are

codes that do not follow these rules. Figure 3-5 shows this:

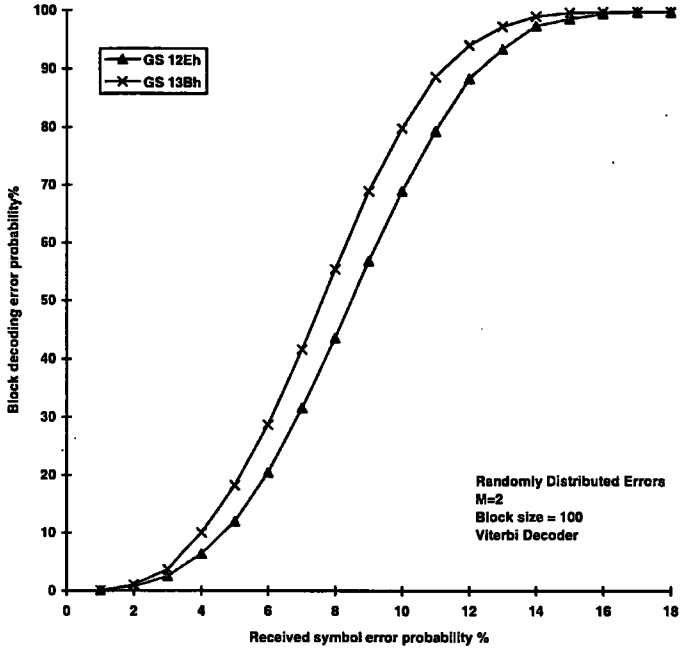


Figure 3-5 **Performance of some other generator sequences.**

GS $12E_h$ = 100 101 110
GS $13B_h$ = 100 111 011

Weight	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GS=														
$12E_h$	0	0	0	0	1	0	0	2	1	2	4	3	7	10
$13B_h$	0	0	0	0	0	2	0	1	0	5	0	9	0	18

Table 3-2 **Distance spectra for some other generator sequences.**

Figure 3-5 shows the superior performance of GS 12Eh over GS $13B_h$ but table 3-2 shows that GS $13B_h$ has a 'better' distance spectrum than GS $12E_h$. Closer examination of table 3-2 reveals the problem; the generator sequences have been sorted and ranked based only on the most

significant spectral component that makes them unique. This is all right if the error probability is small and the minimum free distance term dominates. But when the error probability is large, consideration of other spectral components is necessary.

Figure 3-6 shows the calculated performance of GS $13B_h$ and GS $12E_h$ for error rates between 0.1% and 1.0%. For these low error probability values the performance agrees with the spectral ranking. Figure 3-7 shows the calculated performance for error rates between 1.0% and 18.0%. The y-axis of this graph has been scaled so that the largest value is equal to unity. The reason for this scaling is because the calculation is based on an upper bound which exceeds unity when the symbol error probability is high. This also explains the shape of the graph when compared to the actual performance measured. As can be seen the performance of GS$12E_h$ exceeds the performance of GS$13b_h$ at these higher symbol error probabilities as was seen in figure 3-5.

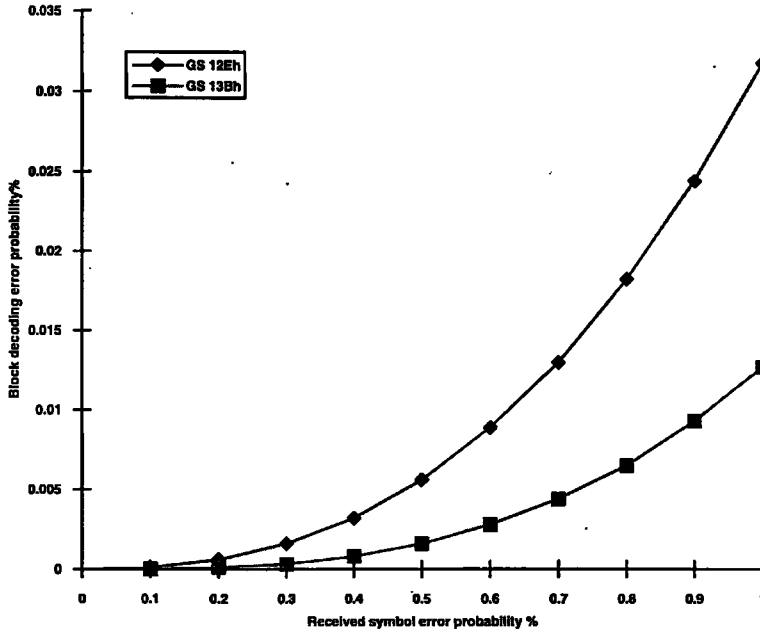


Figure 3-6 Calculated performance of GS $13B_h$ and GS $12E_h$ for small p.

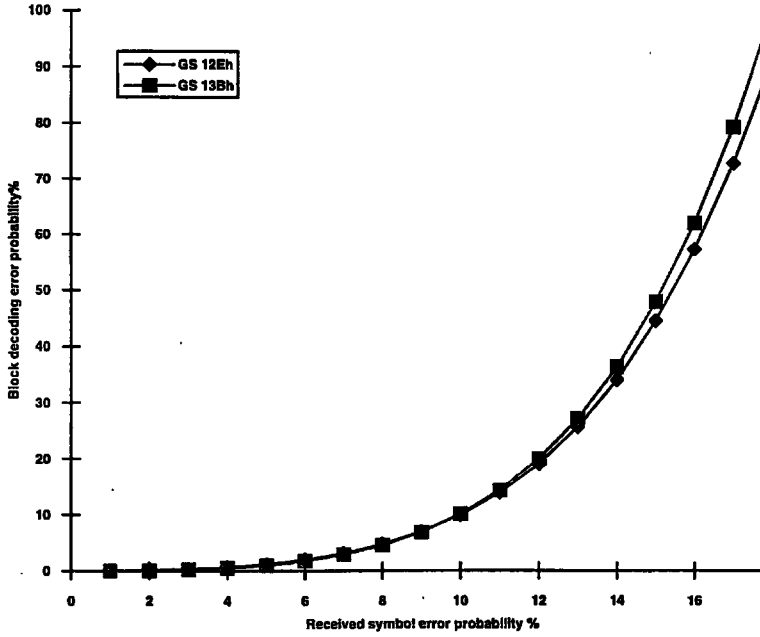


Figure 3-7 Calculated 'scaled' performance of GS $13B_h$ and GS $12E_h$ for large p.

3.6 Good Generator Sequences For Larger *m*.

All of the data obtained from the tree searches was collated and examined. The objective was to determine whether a good generator sequence for a code with alphabet of say 2 was also a good generator sequence for a code with alphabet of say 6.

Below are listed the best 6 generator sequences, together with their distance spectra, for each size of alphabet.

Q=2

Weight	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GS =														
$1AF_h$	00	00	00	00	00	00	01	01	01	02	03	04	06	10
$0EF_h$	00	00	00	00	00	00	01	01	01	02	03	04	06	10
$15F_h$	00	00	00	00	00	00	01	01	01	02	03	04	06	10
$1FD_h$	00	00	00	00	00	00	00	02	00	05	00	13	00	34
$1EF_h$	00	00	00	00	00	00	00	02	00	05	00	13	00	34
$17F_h$	00	00	00	00	00	00	00	02	00	05	00	13	00	34

Q=3

Weight	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GS =														
$1AF_h$	00	00	00	00	00	00	02	02	00	04	12	06	26	42
$0EF_h$	00	00	00	00	00	00	02	02	00	04	12	06	26	42
$15F_h$	00	00	00	00	00	00	02	02	00	04	12	06	26	42
$1FD_h$	00	00	00	00	00	00	00	04	00	08	02	24	24	74
$1EF_h$	00	00	00	00	00	00	00	04	00	08	02	24	24	74
$17F_h$	00	00	00	00	00	00	00	04	00	08	02	24	24	74

Q = 4

Weight	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GS =														
$1AF_h$	00	00	00	00	00	00	03	03	01	06	19	22	54	112
$0EF_h$	00	00	00	00	00	00	03	03	01	06	19	22	54	112
$15F_h$	00	00	00	00	00	00	03	03	01	06	19	22	54	112
$1FD_h$	00	00	00	00	00	00	00	06	00	13	02	41	50	146
$1EF_h$	00	00	00	00	00	00	00	06	00	13	02	41	50	146
$17F_h$	00	00	00	00	00	00	00	06	00	13	02	41	50	146

Q = 5

Weight	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GS =														
$1AF_h$	00	00	00	00	00	00	04	04	00	08	32	36	088	208
$0EF_h$	00	00	00	00	00	00	04	04	00	08	32	36	088	208
$15F_h$	00	00	00	00	00	00	04	04	00	08	32	36	088	208
$1FD_h$	00	00	00	00	00	00	00	08	00	16	04	56	100	224
$1EF_h$	00	00	00	00	00	00	00	08	00	16	04	56	100	224
$17F_h$	00	00	00	00	00	00	00	08	00	16	04	56	100	224

Q = 6

weight	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GS =														
$1AF_h$	00	00	00	00	00	00	05	05	01	10	43	64	144	344
$0EF_h$	00	00	00	00	00	00	05	05	01	10	43	64	144	344
$15F_h$	00	00	00	00	00	00	05	05	01	10	43	64	144	344
$1FD_h$	00	00	00	00	00	00	00	10	00	21	04	77	158	328
$1EF_h$	00	00	00	00	00	00	00	10	00	21	04	77	158	328
$17F_h$	00	00	00	00	00	00	00	10	00	21	04	77	158	328

As can be seen from the above results all the generator sequences are ranked in exactly the same order irrespective of the alphabet size used to generate them. This correspondence does occasionally break down due to the fact that a generator sequence may be catastrophic for one alphabet size but not for another. However, if the generator sequences tried are limited to a common subset in which all those generating catastrophic codes (for any alphabet size) are eliminated then the ranking of them is identical for all alphabet sizes. This result can then be used to find good generator sequences for any alphabet size. To do this, simply rank the generator sequences for a binary alphabet and choose the best of these that is not catastrophic for the required alphabet. This approach makes it relatively quick and easy to find a generator sequence for a code with both a large memory and a large alphabet.

3.7 Chapter Summary.

In this chapter we have looked at the development of a variable rate convolutional encoding/decoding scheme. The scheme may be applied to any decoding method (e.g. Viterbi MLD or Fano sequential decoder) and the performance of the scheme may be derived from the performance of the individual fixed rate schemes. To assess the performance of fixed rate schemes we used the distance spectra of various codes which was obtained

using the FAST algorithm. We saw that the performance of a code can be quickly assessed by looking at the code's minimum free distance provided that the symbol error probability is low. When the symbol error probability is large more care is needed to ensure that the best code is selected and this requires study of the less significant spectral components.

4. Sequential Decoding -Vs- Maximum Likelihood Decoding.

4.1 Introduction.

The Viterbi decoder discussed in appendix B is a popular choice of decoder for decoding convolutional codes because its decoding performance in terms of error correcting power is the best available for a given generator sequence. However, the problem with this decoding approach is that it requires a great deal of memory and is extremely slow compared to alternative decoding strategies. Many efforts have been taken to increase the speed and reduce the memory requirements including combining multiple steps in the decoding tree to reduce the trellis complexity (Wen, Wen & Wang 1990), sending known bits periodically (which are effectively sent for free) reducing the number of states in the decoding trellis (Collins & Hizlan 1993), reorganising the decoding process to decode the error sequence which enables significant pruning of the decoding tree (Ping, Yan & Feng 1991) and paralleling operations to reduce the number of Add-Compare-Select operations (Fettweis & Meyr 1989). Each of these modifications has its advantages but the overall complexity of the decoding procedure (in terms of sophistication) is always increased.

An attractive alternative to the Viterbi decoder is the Fano Sequential decoder discussed in appendix C.

Although the Fano decoder is sub-optimum compared to the Viterbi decoder it has the advantages of being very fast and requiring far less storage. To overcome its sub-optimality it is possible to increase the memory size of the code so that the performance of the Fano decoder is increased to the required level. This increase in code memory size has no effect on the average number of computations required to decode a block of data which significantly increases its effectiveness. One drawback of the Fano sequential decoder is that the actual time taken to decode a given block of data is a random variable. Consequently it is possible for the decoder to take a long time (longer than the Viterbi decoder) to decode some blocks. However, once the time taken has extended beyond a given threshold it is highly likely that the block will end up being decoded in error (Kallel 1988). Because of this, once the threshold is exceeded decoding is abandoned and an error block is output. The rate at which sequential decoding can reliably decode data is set by the so called computational cut-off rate R_{comp} also discussed in appendix C. As with Viterbi MLD efforts have been taken to overcome this disadvantage and it has been shown that R_{comp} can be increased in some circumstances (Kallel 1988).

4.2 Software Development.

To enable a comparison between Viterbi decoding and Fano decoding to be made it was necessary to develop suitable software for this purpose. A suitable Viterbi decoder had already been developed to obtain the results for chapter 3. The software developed for this section was a Fano decoder and some simple routines to allow the results to be analysed. The testing of the Fano decoder was more involved than the Viterbi decoder due to its dynamic nature. The basic difference between the Viterbi and Fano applications was just the decoding routines themselves which had very similar calling parameters.

To gather the required data it was necessary to execute each application 5 times with a different number base each time. Each run varied its error percentage from 1% to 18% and for each error percentage a total of 10,000 blocks were decoded to determine the block failure rate. Despite the fact that the Fano decoder can, in most circumstances, decode much more quickly than the Viterbi decoder, execution of the two applications took approximately the same time. The reason for this is that the Fano decoder is very much slower than the Viterbi decoder at the higher error rates.

4.3 Performance Comparisons Of Different Alphabets.

Figure 4-1 shows the performance of the Viterbi decoder for alphabets ranging from 2 through to 6. The generator sequence used to obtain these results was the one selected in chapter 3 (see figure 3-4 and table 3-1) as being the best 'variable rate' generator sequence and was GS $13D_h$ = 100,111,101. It can be seen from the graph that the improvement in performance for the larger alphabets is considerable at first in moving from an alphabet of size 2 to an alphabet of size 3. However, this improvement diminishes as the alphabet size is increased further and the improvement between the size 5 and size 6 alphabets is less significant.

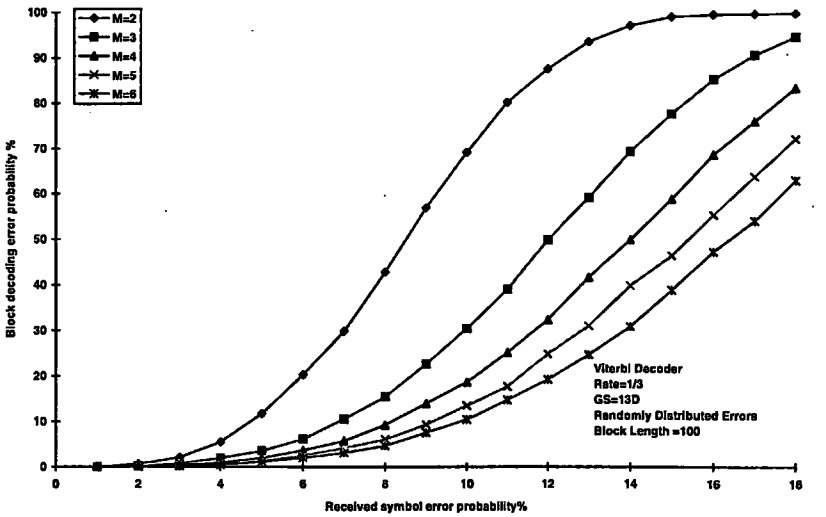


Figure 4-1 **Performance characteristics of Viterbi decoder for various size alphabets.**

Figure 4-2 shows results obtained for the Fano decoder; the overall results are similar to those obtained for the Viterbi decoder except that the performance for each alphabet size is reduced. The reason for this reduction is because of the relatively small block length used. To obtain better performance from the Fano decoder it is necessary to use block sizes upto 10 times longer than those used here. Under these circumstances the difference in performance between the Viterbi decoder and the Fano decoder is quite small.

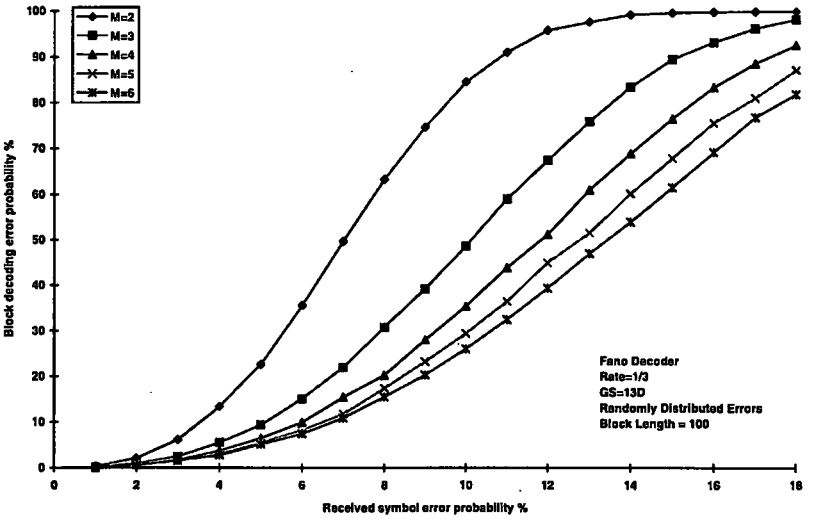


Figure 4-2 **Performance characteristics of Fano decoder for various size alphabets.**

4.4 Improvement Available To Fano Decoder If Code Memory Is Increased.

To emphasise the difference in performance between the two decoders when the code's memory *m* is changed figure 4-3 shows 3 performance characteristics. Two of the characteristics compare the performance of the Viterbi and Fano decoders on the code generated by GS = $13D_h$ = 100,111,101 with an alphabet of size 6. The third characteristic is for a Fano decoder with alphabet of size 6 but now with a generator sequence GS = $20EEF_h$ = 100000,111011,101111. The purpose of showing this is to illustrate that the performance of the Fano decoder can be improved so that it is comparable with the Viterbi decoder using GS = $13D_h$ =100,111,101 by simply increasing the memory order of the code used. As explained in appendix C this increase in memory order does not increase the decoding effort required but the decoding time can be significantly less than the equivalent Viterbi decoder.

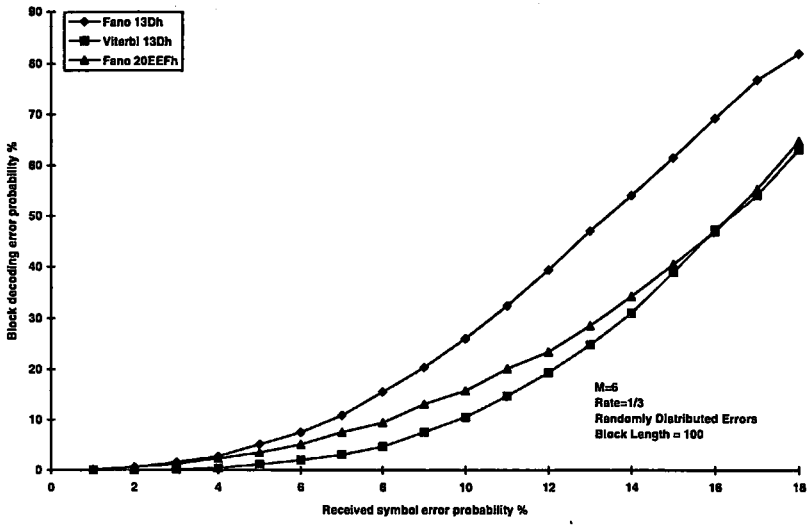


Figure 4-3 **Comparison of code performance showing improvement available to Fano decoder if code memory is increased.**

4.5 Decoding Time.

To illustrate the difference in decoding times for equivalent codes, figures 4-4a and 4-4b show the distribution in decoding times obtained for the Fano decoder working with a code generated by $GS=20EEF_h$ and the time taken for a Viterbi decoder working with a code generated by $GS=13D_h$. The figures show the times taken for those blocks that decoded correctly and those blocks that decoded incorrectly separately. The reason for this is to show that those blocks that decoded correctly did so, on average, in a much shorter time than those that decoded incorrectly. This information can be used to determine a suitable timeout for Fano decoding. In figure 4-4b it is seen that, right upto 14% channel error

rate, all blocks that were decoded correctly were decoded in less than 400ms. In comparison, figure 4-4a shows that blocks with more than 7% channel errors took, on average, longer than 400ms to decode when decoded incorrectly. Therefore, if a timeout of 400ms is applied to the Fano decoder (which happens to be the time the Viterbi decoder requires to decode a similar block of data) the time wasted on blocks that will end up containing errors is reduced. To avoid complicating the graph the blocks that 'timed-out' during the Fano decoding process are not included. These times must be considered to gain a full picture especially when considering higher symbol error rates. It is worth noting that no time-outs occurred in any block corrupted with 6% or fewer errors and even at 10% error rate only 0.15% of blocks timed-out.

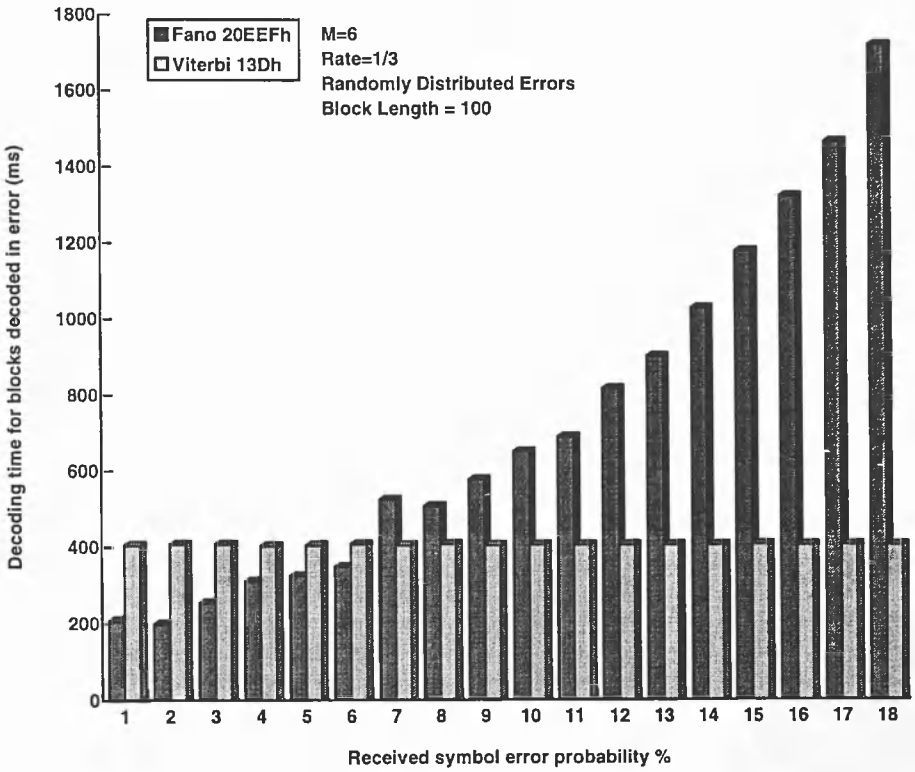


Figure 4-4a Distribution in decoding times of incorrectly decoded blocks for Fano and Viterbi decoders.

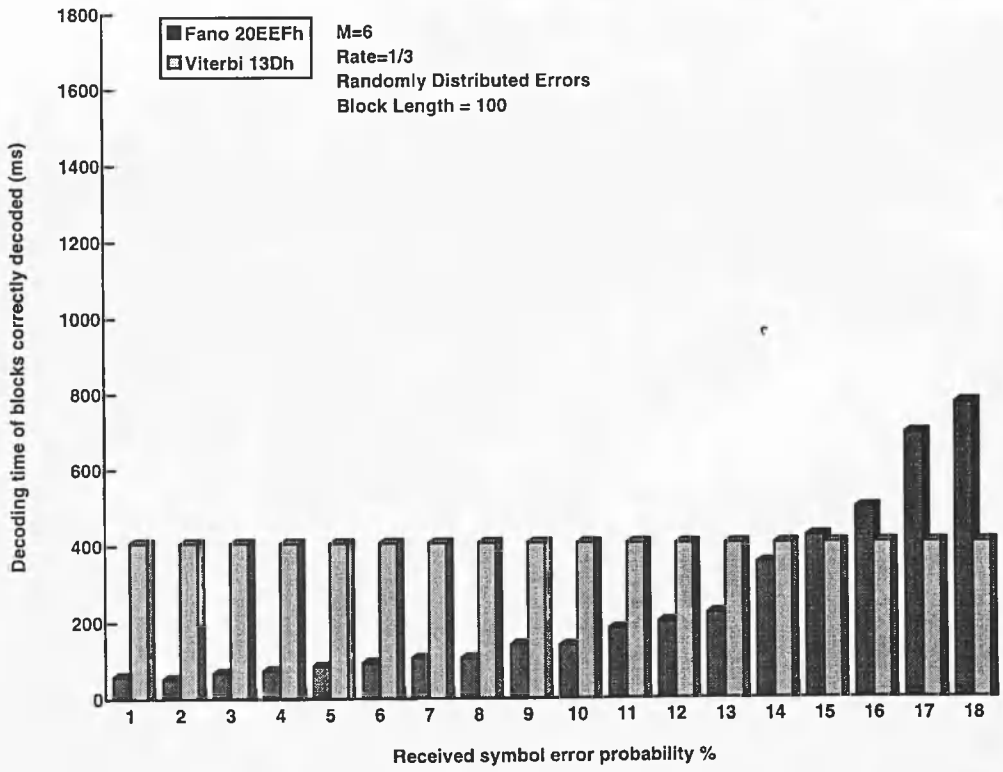


Figure 4-4b Distribution in decoding times of correctly decoded blocks for Fano and Viterbi decoders.

As can be seen the time taken to decode a block of data is constant for the Viterbi decoder at around 400 ms. However, the time taken to decode a block of data using the Fano decoder is both dependent on the number of errors in the received block and upon whether the block ended up being decoded in error or correctly. A fuller picture is obtained by studying tables 4-1a and 4-1b below, where CER represents channel error rate.

CER	Blocks Failed	Average Decode Time	Average Number of Computations
01%	0.13%	207ms	421.54
02%	0.31%	199ms	422.48
03%	0.69%	254ms	531.10
04%	1.48%	311ms	656.79
05%	2.52%	324ms	680.91
06%	3.56%	348ms	732.64
07%	6.47%	524ms	1099.66
08%	8.30%	508ms	1068.41
09%	10.49%	557ms	1172.71
10%	12.90%	649ms	1361.43
11%	15.68%	689ms	1590.38
12%	17.62%	818ms	1718.70
13%	20.52%	904ms	1898.10
14%	28.23%	1031ms	2327.75
15%	30.79%	1182ms	2566.57
16%	34.63%	1327ms	2815.36
17%	38.20%	1470ms	3343.82
18%	42.40%	1722ms	3677.40

***Table 4-1a* Results for blocks decoding in error using Fano decoder.**

CER	Blocks Passed	Average Decode Time	Average Number of Computations
01%	99.87%	058ms	122.50
02%	99.69%	051ms	132.52
03%	99.31%	067ms	141.81
04%	98.52%	072ms	152.63
05%	97.48%	085ms	180.19
06%	96.44%	093ms	196.63
07%	93.53%	104ms	220.83
08%	91.70%	105ms	248.23
09%	89.51%	139ms	295.24
10%	87.10%	138ms	317.27
11%	84.32%	182ms	384.99
12%	82.38%	199ms	448.14
13%	79.48%	223ms	499.76
14%	71.77%	354ms	745.38
15%	69.21%	424ms	893.79
16%	65.37%	498ms	1122.26
17%	61.80%	695ms	1463.70
18%	57.60%	774ms	1809.05

Table 4-1b **Results for blocks decoding correctly using Fano decoder.**

From these tables it can be seen that the majority of blocks are decoded correctly right up to a channel error rate of 18%. This bias towards correct decoding has the effect of weighting the graphs of figures 4-4a and 4-4b in favour of figure 4-4b. The consequence of this is that Fano decoding has both a memory-requirement advantage and decoding-time advantage over the Viterbi decoder for channel error rates up to about 13%. The overall decoding-time advantage is shown in table 4-2 and figure 4-5.

CER	Decode Time for Viterbi Decoder	Average Decode Time for Fano Decoder
01%	406ms	58ms
02%	406ms	51ms
03%	406ms	68ms
04%	406ms	76ms
05%	406ms	91ms
06%	406ms	102ms
07%	406ms	131ms
08%	406ms	138ms
09%	406ms	182ms
10%	406ms	203ms
11%	406ms	262ms
12%	406ms	308ms
13%	406ms	363ms
14%	406ms	545ms
15%	406ms	657ms
16%	406ms	785ms
17%	406ms	991ms
18%	406ms	1176ms

Table 4-2 **Decoding results for all blocks using Fano and Viterbi decoder.**

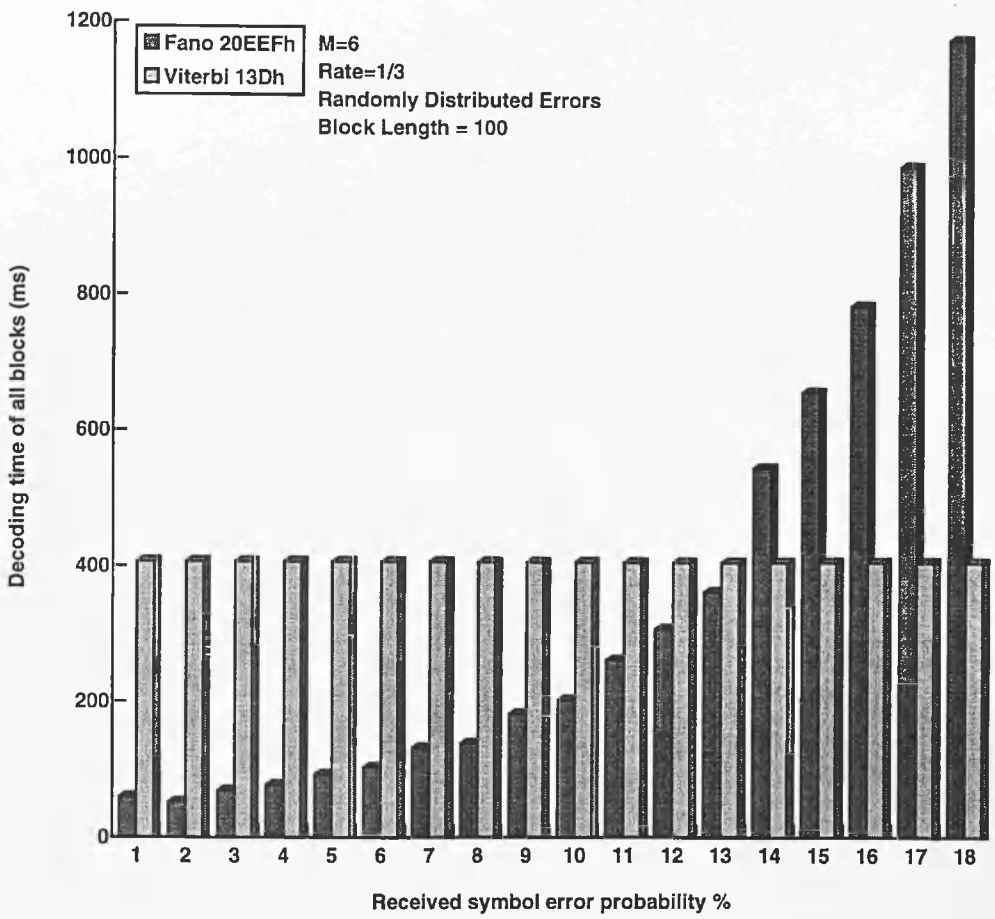


Figure 4-5 **Distribution in decoding times for all blocks for Fano and Viterbi decoders.**

As shall be seen in the next section it is rare for the CER to exceed 10% and a more normal figure - even when symbol periods are short - is around 6%.

In tables 4-1a and 4-1b the right-hand column indicates the 'Average Number of Computations' required to decode a block of data where a computation is defined as the metric calculation of a path extension. When the same definition was used for the Viterbi decoder the number of computations required to decode a block of data was (block size)*(6^3) = 115*216 = 24840. Referring to table 4-1b and the row corresponding to 18% CER it can be seen that the average number of computations here is around 3677 which is less than 1/6 the computations required by the Viterbi decoder. However, as figure 4-4a makes clear, the actual time taken to decode using the Viterbi decoder is less than 1/4 that required by the Fano decoder. It is clear from this that 'computations' as defined above is not an appropriate measure for comparing performance and that actual decoding time is a better measure. This discrepancy is due to the additional 'processing' required in the software to execute the Fano algorithm - features such as back-tracking, setting and clearing of flags, choosing the next branch for exploration, etc. all carry a computational overhead. As a general comment the software to execute each decoder used many common subroutines and the use of look-up

tables were used wherever possible to speed up execution and to keep comparisons fair.

4.6 Chapter Summary.

In this chapter we have looked at the advantages and disadvantages of both the Viterbi and Fano decoder. It was shown that the resources required by the Viterbi decoder are very much greater than the Fano decoder and that the sub-optimality of the Fano decoder can be overcome by increasing the code's memory. Increasing the code's memory does not effect the Fano decoder in terms of computations required but does bring its performance in-line with that of the 'equivalent' Viterbi decoder. Finally we looked at the advantages the Fano decoder can provide over and above the Viterbi decoder by presenting simulation data.

5. Throughput Analysis.

5.1 Introduction.

In this section the throughput of the scheme developed will be analysed and compared with fixed rate FEC schemes and standard ARQ. The analysis will focus on the throughput for two types of channel:

- A simple random error channel with channel error rates varying from 1% to 18% as in previous chapters.
- Channels constructed using error maps. The error maps are obtained from recorded data sent over real HF channels.

The objective of the analysis is to show that the variable rate scheme is able to transfer a given amount of user data over the selected channel using significantly fewer blocks than either a fixed rate FEC or ARQ scheme. This will be shown in a number of ways:

- An analysis of the individual rate schemes (i.e. 1/1 1/2 rate and 1/3 rate) is provided based on the results obtained from the raw error-correcting power at the various rates.
- Using the simple channel with error rates varying between 1% and 18% it is shown using simulation that the throughput of the variable rate scheme tracks the

upper envelope of the individual throughputs for each of the fixed rates.

- To show the relative throughput for the simulated HF channels the number of blocks required to transmit a given 'file' of user data is counted and compared with the number required for the fixed rate FEC schemes and the ARQ scheme.

5.2 Software Development.

The software developed for this chapter was that necessary to allow the throughput of various schemes to be evaluated. The decoder used was the Fano decoder developed for the analysis performed in chapter 4. To allow various fixed rate decoding schemes, as well as variable rate decoding to be analysed, some modification of the software was necessary. The analysis of error distributions within blocks of data called for new software to be written. However, compared to the decoding applications, this software was simple.

5.3 Throughput Analysis Of Variable Rate Scheme Using A Simple Channel Model.

The block decoding performance for the various individual rates of the Fano decoder used with a code generated by GS = 20EEF is given in table 5-1. This is then shown as throughput efficiency, by taking account of the inherent reduction in throughput for the 1/2 and 1/3 rate codes, in table 5-2 and figure 5-1.

CER%	Pass rate (%) for 1:1 code	Pass rate (%) for 1:2 code	Pass rate (%) for 1:3 code
00	100	100	100
01	0	98.67	99.84
02	0	93.48	99.33
03	0	86.26	98.78
04	0	76.82	97.63
05	0	67.00	96.47
06	0	55.95	94.85
07	0	44.82	92.39
08	0	35.46	90.52
09	0	28.15	86.92
10	0	20.31	84.22
11	0	14.66	79.91
12	0	9.80	76.57
13	0	6.68	71.51
14	0	4.36	65.81
15	0	2.60	59.53
16	0	1.48	53.12
17	0	0.79	44.67
18	0	0.41	35.21

Table 5-1 **Block decoding performance of individual rates using code generated by GS = 20EEF.**

CER%	Throughput Efficiency(%) for 1:1 code	Throughput Efficiency(%) for 1:2 code	Throughput Efficiency(%) for 1:3 code
00	100	50	33.33
01	0	49.34	33.28
02	0	46.74	33.11
03	0	43.13	32.93
04	0	38.41	32.54
05	0	33.50	32.16
06	0	27.98	31.62
07	0	22.41	30.80
08	0	17.73	30.17
09	0	14.08	28.97
10	0	10.16	28.07
11	0	7.33	26.64
12	0	4.90	25.52
13	0	3.34	23.84
14	0	2.18	21.94
15	0	1.30	19.84
16	0	0.74	17.71
17	0	0.40	14.89
18	0	0.21	11.74

Table 5-2 **Throughput efficiency of individual rates using code generated by GS = 20EEF.**

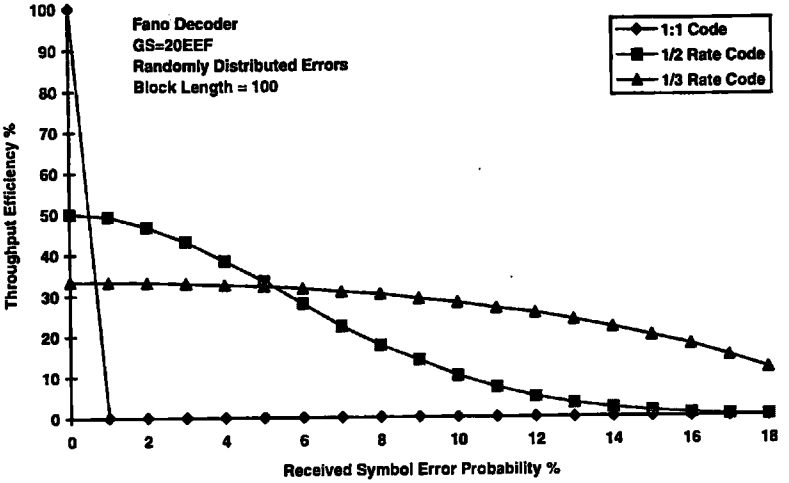
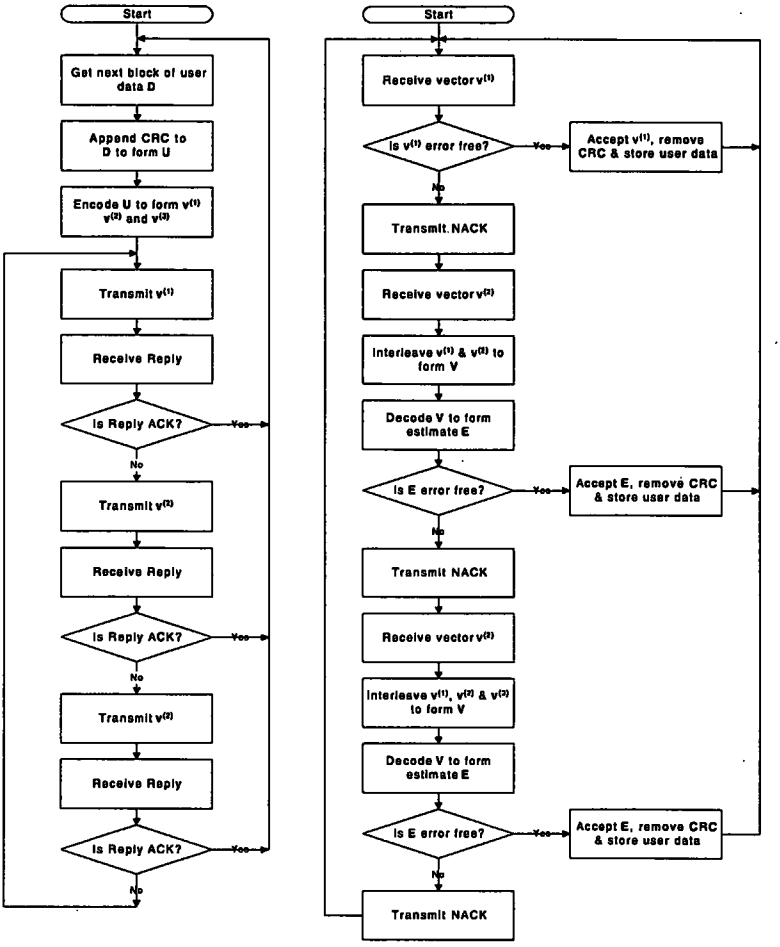


Figure 5-1 **Throughput efficiency of individual rates using code generated by GS = 20EEF.**

If we take the upper envelope of these three characteristics it is possible to compare the throughput of the variable rate scheme against the best throughput available from any of the individual rate schemes. To determine the throughput of the variable rate scheme the software was configured as described in chapter 3 section 3.4 and the flow chart describing this is reproduced below in figure 5-2.



(a) Transmit Site (b) Receive Site

***Figure 5-2* Flow chart of variable rate scheme.**

The throughput was determined by allowing 10,000 blocks to be sent over the simulated channel and counting how many user data blocks were received correctly. The throughput then was simply the ratio of user data blocks received to channel blocks transmitted. The results obtained are shown in figure 5-3.

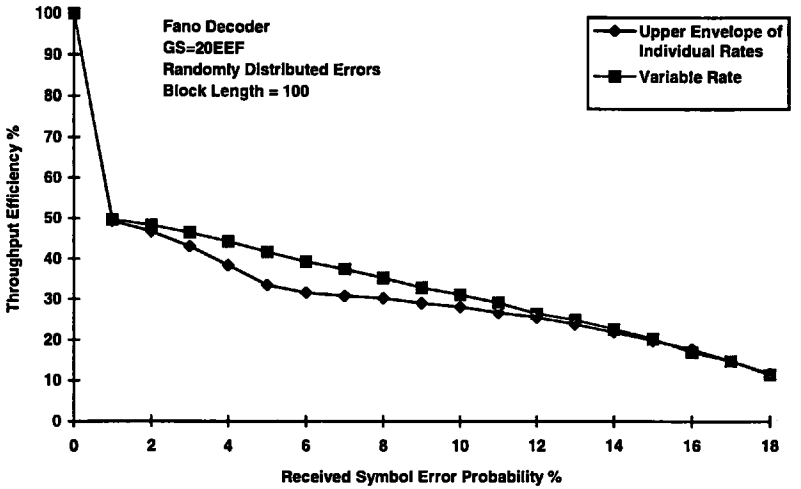


Figure 5-3 **Throughput achieved using variable rate.**

As can be seen from figure 5-3, the throughput achieved by the variable rate scheme closely tracks the upper envelope of the individual rates and at channel error rates around 6% the throughput is better than the upper envelope. The significance of this is discussed below.

5.4 Throughput Analysis For The HF Channel.

The HF channel has time varying characteristics that cause it to be virtually error free during some periods and virtually unusable during other periods. Fortunately, if care is taken to choose suitable frequencies, the proportion of the time over which the channel is unusable can be made quite small. There have been many efforts to characterise the HF channel and many complex models are available to predict which frequencies

should be used at particular times of the day, month and year.

5.4.1 HF Channel Models & Error Distributions.

Rather than use a *model* of the HF channel for this section, error maps constructed from real HF transmissions were used. The error maps were constructed over a 1-month period whilst conducting field trials. One site was in England, the other in Central America. At the UK site good transmit and receive antennas were available and high transmission powers were used. At the site in Central America simple whip antennas were used and transmit power was limited to 25W. The modulation scheme used was Orthogonal 6-tone Multi-Frequency Shift Keying at rates varying from 75 bits/sec to 600 bits/sec where 2 6-tone symbols were used to represented 5 bits of data i.e. each character from a 32 character alphabet was mapped to a 5-bit word and transmitted using 2 6-tone symbols.

To construct the error maps large files of known data were transmitted and compared at the receive site with copies of the known data. At each location where an error occurred a note was made of:

i) the location

ii) the received symbol

iii) the known symbol

Later, when all the data was analysed, particular attention was paid to the corruptions to see if there was any relationship between the received symbol and the known symbol. In particular, the data was examined to see if a corrupted tone was more likely to be corrupted to one of its neighbours than to any other tone. The analysis showed that the distribution of errors was random and that there was no relationship between received tones and known tones. This simplified matters as the error map could now consist of 1's and 0's: a 1 to represent an error and a 0 to represent no error. To simulate the HF channel, as each tone is 'transmitted' the error map is examined. If the error map indicated no error then the 'transmitted' symbol is unaltered. If the error map indicated that an error should be introduced then the 'transmitted' symbol is corrupted to some other random symbol (care being taken to ensure that the random symbol is different to the one being 'transmitted').

As the error maps are to be used with an error correcting scheme that transmits data in blocks of 100 symbols it is interesting to know how errors are distributed amongst these blocks. The error distributions obtained from each of the maps are shown below in tables 5-3 to 5-6 and in figures 5-4 to 5-7. Figures 5-4 to 5-7 are split into two: (a) and (b). (a) shows the distribution of blocks around the average error rate, where most blocks are

concentrated. (b) shows the block distribution over the whole range but with the y-axis limited so that the detail can be seen.

N	Blocks With N Errors	N	Blocks With N Errors	N	Blocks With N Errors	N	Blocks With N Errors
00	3915	26	2	52	0	78	0
01	236	27	3	53	0	79	2
02	335	28	1	54	2	80	1
03	113	29	4	55	0	81	0
04	104	30	3	56	0	82	0
05	60	31	0	57	0	83	0
06	51	32	2	58	0	84	0
07	32	33	0	59	0	85	0
08	31	34	0	60	0	86	0
09	19	35	0	61	0	87	0
10	11	36	0	62	0	88	0
11	8	37	0	63	0	89	0
12	17	38	1	64	0	90	0
13	6	39	1	65	0	91	0
14	4	40	1	66	0	92	0
15	5	41	0	67	0	93	0
16	7	42	0	68	0	94	0
17	3	43	0	69	0	95	0
18	3	44	0	70	1	96	0
19	4	45	0	71	1	97	0
20	5	46	0	72	0	98	0
21	1	47	0	73	0	99	0
22	5	48	0	74	0	100	0
23	3	49	0	75	0		
24	3	50	0	76	0		
25	2	51	1	77	0		

Table 5-3 **Error distributions at 75 bits/sec.**

Distribution statistics:

Total Errors = 5322
Total Symbols = 500900
Average Error% = 1.0%

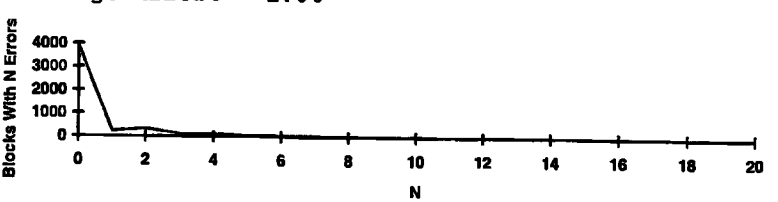


Figure 5-4a **Block distribution around average error% at 75 bits/sec.**

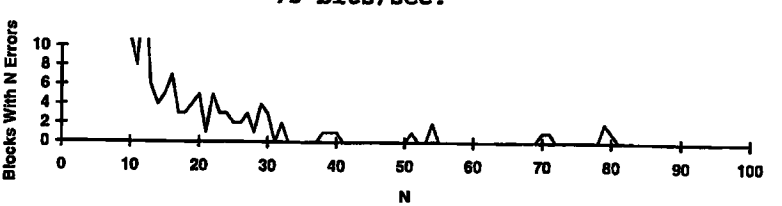


Figure 5-4b **Overall block distribution at 75 bits/sec.**

N	Blocks With N Errors	N	Blocks With N Errors	N	Blocks With N Errors	N	Blocks With N Errors
00	7926	26	8	52	3	78	0
01	435	27	8	53	3	79	0
02	586	28	4	54	1	80	0
03	230	29	3	55	3	81	0
04	219	30	4	56	2	82	0
05	117	31	1	57	2	83	0
06	117	32	2	58	1	84	0
07	71	33	4	59	0	85	0
08	67	34	5	60	0	86	0
09	56	35	3	61	5	87	0
10	36	36	2	62	1	88	0
11	30	37	5	63	0	89	0
12	23	38	2	64	3	90	0
13	23	39	1	65	3	91	0
14	22	40	3	66	0	92	0
15	12	41	7	67	0	93	0
16	20	42	2	68	0	94	0
17	13	43	1	69	0	95	0
18	15	44	1	70	3	96	0
19	18	45	2	71	0	97	0
20	15	46	1	72	2	98	0
21	9	47	3	73	1	99	0
22	5	48	2	74	0	100	0
23	12	49	1	75	0		
24	10	50	3	76	0		
25	5	51	1	77	0		

Table 5-4 **Error distributions at 150 bits/sec.**

Distribution statistics:

Total Errors = 14960
Total Symbols = 1020400
Average Error% = 1.4%

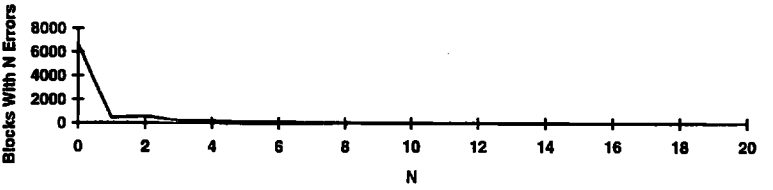


Figure 5-5a **Block distribution around average error% at 150 bits/sec.**

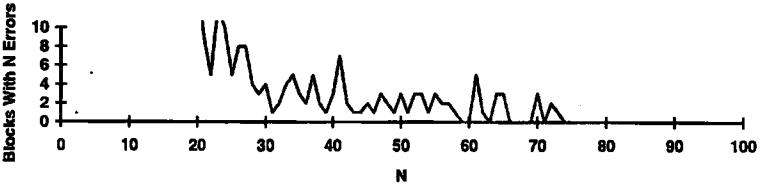


Figure 5-5b **Overall block distribution at 150 bits/sec.**

N	Blocks With N Errors	N	Blocks With N Errors	N	Blocks With N Errors	N	Blocks With N Errors
00	13618	26	27	52	8	78	3
01	974	27	26	53	4	79	2
02	1260	28	31	54	1	80	4
03	679	29	16	55	3	81	6
04	611	30	21	56	1	82	2
05	490	31	22	57	4	83	4
06	421	32	15	58	3	84	3
07	327	33	12	59	0	85	3
08	242	34	13	60	4	86	1
09	199	35	7	61	2	87	6
10	194	36	11	62	3	88	6
11	173	37	15	63	1	89	1
12	138	38	18	64	4	90	1
13	101	39	9	65	1	91	1
14	98	40	11	66	1	92	1
15	89	41	8	67	3	93	1
16	90	42	12	68	2	94	0
17	66	43	2	69	4	95	0
18	61	44	6	70	1	96	0
19	80	45	6	71	2	97	0
20	50	46	4	72	4	98	0
21	44	47	7	73	5	99	0
22	32	48	3	74	4	100	0
23	33	49	2	75	0		
24	32	50	4	76	4		
25	27	51	0	77	0		

***Table 5-5* Error distributions at 300 bits/sec.**

Distribution statistics:

Total Errors = 57113
Total Symbols = 2055100
Average Error% = 2.7%

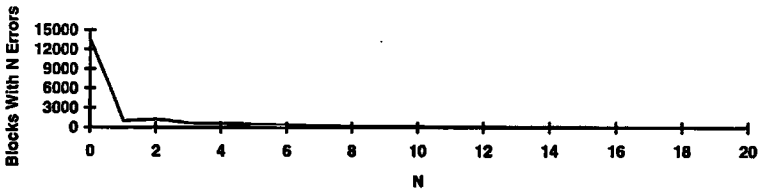


***Figure 5-6a* Block distribution around average error% at 300 bits/sec.**

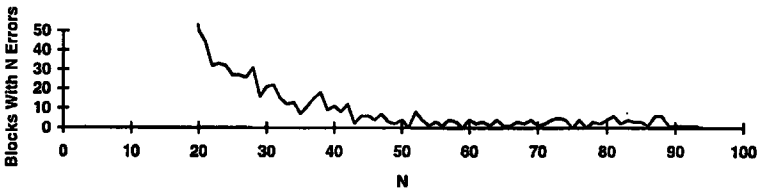


***Figure 5-6b* Overall block distribution at 300 bits/sec.**

N	Blocks With N Errors	N	Blocks With N Errors	N	Blocks With N Errors	N	Blocks With N Errors
00	18558	26	145	52	26	78	18
01	1210	27	120	53	30	79	12
02	1643	28	132	54	23	80	10
03	1099	29	126	55	13	81	09
04	1191	30	114	56	17	82	10
05	927	31	121	57	17	83	13
06	952	32	115	58	22	84	4
07	806	33	89	59	15	85	5
08	744	34	67	60	16	86	10
09	668	35	90	61	17	87	5
10	590	36	82	62	16	88	5
11	596	37	71	63	22	89	2
12	512	38	68	64	8	90	0
13	452	39	56	65	17	91	3
14	441	40	56	66	14	92	1
15	358	41	58	67	12	93	1
16	376	42	39	68	8	94	1
17	341	43	48	69	19	95	0
18	328	44	43	70	12	96	0
19	282	45	40	71	8	97	0
20	276	46	47	72	11	98	0
21	255	47	34	73	8	99	0
22	186	48	33	74	9	100	0
23	190	49	49	75	9		
24	196	50	33	76	9		
25	184	51	23	77	12		

***Table 5-6* Error distributions at 600 bits/sec.**

Distribution statistics:

Total Errors = 228044
Total Symbols = 3575900
Average Error% = 6.3%

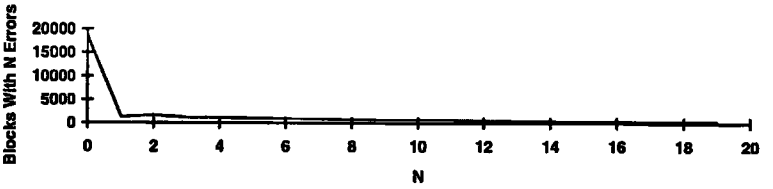


***Figure 5-7a* Block distribution around average error% at 600 bits/sec.**

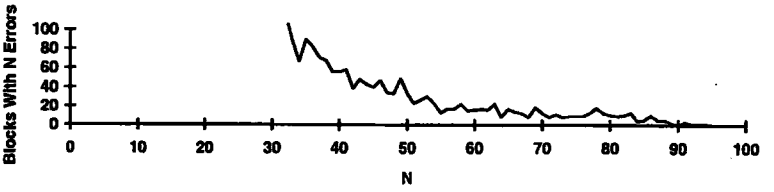


Figure 5-7b Overall block distribution at 600 bits/sec.

5.4.2 HF Channel Results.

From the above tables and figures it is clear that:

- A very large number of transmitted blocks have no errors.
- Only a very small proportion of transmitted blocks have more than around 18% errors, even at the higher transmission rates when channel errors are more likely.

It is these two points that make the variable rate scheme so effective on the HF channel:

- The scheme first looks at the block to see if it is error free and if it is accepts it without the need to perform decoding. This will happen frequently as we have seen and under such circumstances the throughput will approach 100% (it is less than 100% because of the CRC symbols).
- Even when the channel is noisy and the data rate is at its highest the 1/3 rate code is, in the vast majority of cases, able to keep the throughput at around 20%. Thus, extending the scheme to include a 1/4 rate code would give negligible benefit.
- Also, at the highest transmission rate, the average error probability of channel symbols was around 6%. Figure 5-3 shows that the throughput of the variable rate scheme is better than any of the individual fixed rate schemes in this region further improving the scheme's effectiveness over the HF channel.

To demonstrate the effectiveness of the variable rate scheme the program was executed using the error maps described above to determine the error locations. The average throughput in each case was recorded.

Next a simple ARQ scheme was executed using the same error maps and again the throughput in each case recorded.

Finally two fixed rate FEC schemes were executed (1/2 rate and 1/3 rate) using the error maps and the throughputs recorded.

Table 5-7 and figure 5-8 show the results obtained.

	Variable	ARQ	1/2 Rate FEC	1/3 Rate FEC
75 Bits/sec	59.24	58.59	32.42	23.98
150 Bits/sec	117.50	116.75	64.43	47.76
300 Bits/sec	204.12	194.61	112.38	90.12
600 Bits/sec	325.62	292.62	174.72	156.96

Table 5-7 **Throughput (in bits/sec.) of various schemes at various rates.**

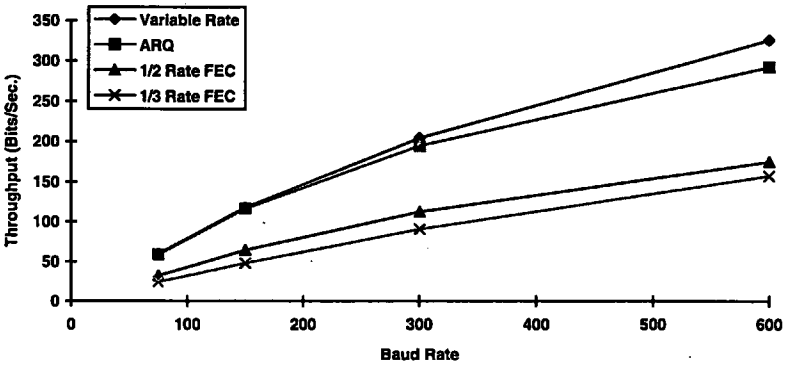


Figure 5-8 **Throughput of various schemes at various rates.**

As can be seen from figure 5-8 the throughput of all schemes is highest at the highest transmission rate (600 bits/sec). This is because the degradation in

demodulator performance due to the reduced symbol duration is more than compensated for by the higher transmission rate. Also, as was expected, the throughput of the variable rate scheme is higher than any of the other schemes. A surprising result is the fact that the simple ARQ scheme out-performs either of the fixed rate FEC schemes. This consolidates the view that the HF channel is able to support error free communication for a large proportion of the time even at the higher data rates.

5.5 Interleaving.

The fact that the HF channel produces errors in bursts is detrimental to the error correcting performance of the scheme developed. The likelihood of correct decoding is increased if the errors are uniformly distributed throughout the block rather than grouped together in clusters. A technique commonly used to break-up these error bursts is interleaving where the data to be transmitted is written into an array in rows and then read-out in columns as the data is transmitted. The transmitted data is then subjected to the error bursts as usual. At the receive end of the link the channel data is written into the columns of an identical array and then read-out in rows prior to decoding. The net effect of this process is that the order of data remains unaltered but that the error bursts are broken up. The variable rate scheme has interleaving built-in due to the

transmission of the encoder vectors in different time frames. However, the technique described above was applied using an array of dimension 5 x 23, giving an interleave depth of 5, to see if any extra benifit could be obtained from this second level of interleaving. The throughput improvement is shown in table 5-8 and figure 5-9.

	Variable	ARQ	1/2 Rate FEC	1/3 Rate FEC
75 Bits/sec	61.62	58.59	36.13	24.49
150 Bits/sec	121.84	116.75	71.69	48.27
300 Bits/sec	216.98	194.61	132.12	91.47
600 Bits/sec	363.02	292.62	222.24	161.64

***Table 5-8* Throughput improvement using interleaving.**

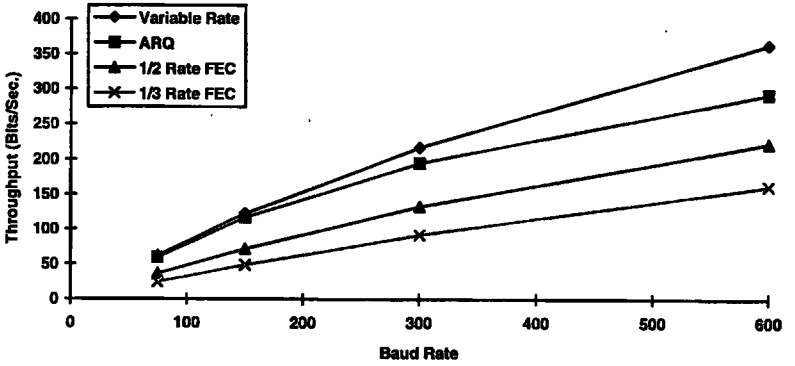


***Figure 5-9* Throughput improvement using interleaving.**

Comparing figures 5-8 and 5-9 it can be seen that the ranking order of the various schemes are the same both with and without interleaving. The actual improvement available from interleaving can be seen by comparing Tables 5-7 and 5-8. In every case, with the exception of the ARQ scheme, interleaving provides an increase in throughput. At 600 bits/sec the throughput of the

variable rate scheme is increased from 329.7 bits/sec to 362.34 bits/sec which is almost 10%.

In the above graphs it is clear that the performance of the ARQ scheme is almost as good as the variable rate scheme. This is because of the large percentage of blocks that are error free. A minor modification to the decoding algorithm is able to take further advantage of this fact. If the code is chosen so that each vector received by the decoding scheme is invertible (i.e. if it is error free the original information sequence can be derived from it) then before interleaving and subsequent decoding takes place the vector itself can be tested to see if the original data can be retrieved. When the previous vector or vectors contain many errors it is possible that even an error free transmission of the next vector will not reduce the overall error count sufficiently to allow correct decoding to take place. Under these conditions such an approach will improve throughput.

For each vector to be invertible the generator sequence of the code must have the property that the 1st stage in the encoder shift register is connected to each of the modulo-M adders forming the output vectors. It turns out that this requirement is satisfied by many good generator sequences. Take as an example the generator sequence $20EEF_h$ used in this chapter:

$$20EEF_h = 100000\ 111011\ 101111$$

so that:

$$v_i^{(1)} = u_i$$

$$v_i^{(2)} = u_i + u_{i-1} + u_{i-2} + u_{i-4} + u_{i-5}$$

$$v_i^{(3)} = u_i + u_{i-2} + u_{i-3} + u_{i-4} + u_{i-5}$$

rearranging these equations we have:

$$u_i = v_i^{(1)}$$

$$u_i = v_i^{(2)} - u_{i-1} - u_{i-2} - u_{i-4} - u_{i-5}$$

$$u_i = v_i^{(3)} = - u_{i-2} - u_{i-3} - u_{i-4} - u_{i-5}$$

So as decoding proceeds provided we know u_i at time i then we also know u_{i-t} at time t as this was the output t time units before.

This modification was made to the scheme and the results obtained are shown in table 5-9. The conditions under which these results were obtained are identical to those in table 5-8.

	Modified Variable	Variable	ARQ
75 Bits/sec	62.81	61.62	58.59
150 Bits/sec	124.11	121.84	116.75
300 Bits/sec	219.21	216.98	194.61
600 Bits/sec	363.18	361.72	292.62

Table 5-9 **Throughput improvement using invertible vectors.**

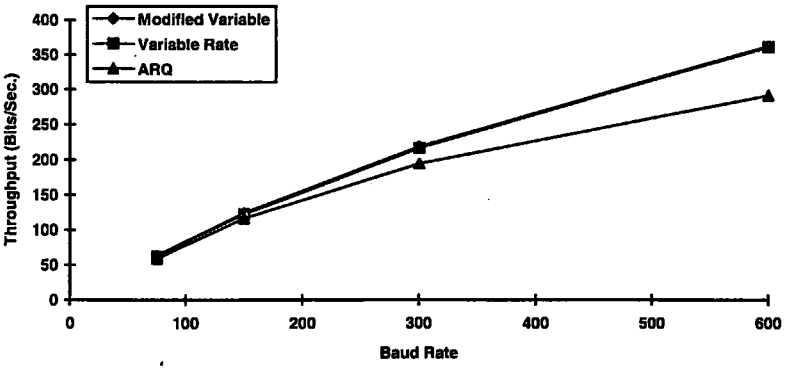


Figure 5-10 **Throughput improvement using invertible vectors.**

As can be seen from table 5-9 and figure 5-10 the throughput improvement obtained using this modified scheme is negligible. However, the modification is easily implemented, does give a slight improvement in throughput and does speed-up decoding.

5.6 Chapter Summary.

In this chapter the throughput of various schemes has been analysed. Initially the block decoding performance of the various fixed rate schemes at channel error rates ranging from 1% to 18% was examined and converted to throughput by taking account of the redundancy in each. Using these throughput curves an upper envelope was defined which represented the best throughput available at the fixed rates. The variable rate scheme was then analysed and its throughput compared with this upper envelope. The results showed that the variable rate scheme tracked the upper envelope closely and in the vicinity of 6% channel error rate the throughput was better.

Next the performance of the various schemes over simulated HF channels was analysed. The simulated channels were constructed using error maps and the error maps were generated using live transmissions at 75 bits/sec, 150 bits/sec, 300 bits/sec and 600 bits/sec. The superior performance of the variable rate scheme compared to the other schemes was demonstrated and the

fact that throughput was highest at the highest transmission rate for all schemes noted.

Next interleaving was discussed and it was shown that the use of interleaving increased the throughput for all schemes except the ARQ scheme.

Finally the scheme was modified so that each vector received by the decoding scheme could be inverted to see if the original data could be retrieved directly from it. The throughput improvement obtained by this modification was negligible.

6. Implementation Issues.

6.1 Introduction.

A hybrid FEC/ARQ scheme used for the transmission and reception of data over the HF channel must be capable of dealing with periods of complete channel loss and should allow a number of unacknowledged blocks to be outstanding. At the transmit site a simple hybrid FEC/ARQ scheme receives a block of data for transmission and encodes it using a convolutional encoder with a given set of parameters. The output of the encoder is stored as three separate 'de-interleaved' vectors prior to transmission as shown in figure 6-1 below.

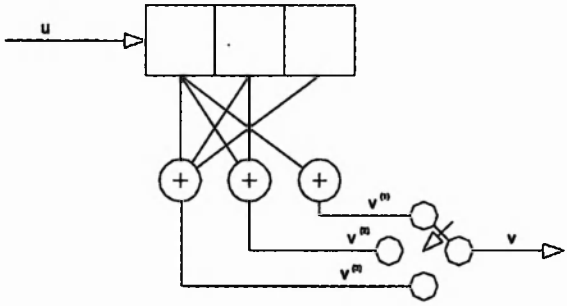


Figure 6-1 **Convolutional encoder output.**

$$U = u_0, u_1, \ldots, u_N.$$

$$V^{(1)} = v_0^{(1)}, v_1^{(1)}, \ldots, v_N^{(1)}.$$

$$V^{(2)} = v_0^{(2)}, v_1^{(2)}, \ldots, v_N^{(2)}.$$

$$V^{(3)} = v_0^{(3)}, v_1^{(3)}, \ldots, v_N^{(3)}.$$

$$V = v_0^{(1)}, v_0^{(2)}, v_0^{(3)}, v_1^{(1)}, v_1^{(2)}, v_1^{(3)}, \ldots v_N^{(1)}, v_N^{(2)}, v_N^{(3)}.$$

The vectors $V^{(1)}$, $V^{(2)}$ and $V^{(3)}$ are the de-interleaved vectors and the one selected for transmission depends on the current state of the encoding/decoding process. At the receive site a simple hybrid FEC/ARQ scheme receives a possibly erroneous version of the 'de-interleaved' vector sent from the transmit site and interleaves it with the appropriate previous versions it holds in its store. When the interleave process is complete the resulting data block is decoded using a convolutional decoder with the appropriate parameters according to the current state of the encoding/decoding process. If decoding is successful the receive site informs the transmit site, via a perfect return channel, that decoding was successful and that the next block of data for transmission should be processed and sent. If decoding is unsuccessful then the receive site requests the transmit site for the next version of the current block so that it may attempt decoding again. In this way the complete message is sent from one end to another as a series of blocks, each block being acknowledged before the next is sent. This scheme is simple to use and works well provided that the *control-data* flowing from receive site to transmit site is guaranteed error free and that the stop/go nature of the protocol can be tolerated.

6.2 Software Development.

Most of the effort for this chapter went into the development of the protocol software and full details of the implementation and complexity of the protocol can be found in Appendix D. The decoder used was the Fano decoder developed in chapter 4 with no modifications. The testing of the software was very time consuming and was, to a large extent, done manually. It would have been possible to write code to test the protocol, but then this code would have to be tested somehow. To assist with the manual testing of the protocol some debugging routines were developed that allowed the data in the system to be examined in a convenient form.

6.3 A Practical Scheme For Use Over A Noisy Channel.

In a practical system there is no guarantee that the control information received by the transmit site is error free. Also, in most situations, there is a need to communicate messages in both directions. Note that we make the distinction between *messages* meaning user data and *control information* which is simply information that the transmit and receive sites must exchange to keep in synchronism with each other. With these points in mind we develop a suitable protocol to support the variable rate scheme.

To deal with the possible corruption of control-data and the ability to have a number of unacknowledged blocks outstanding additional information must be sent with each block transmitted. The block structure at each stage of the prepare/encode cycle is as shown in figure 6-2.

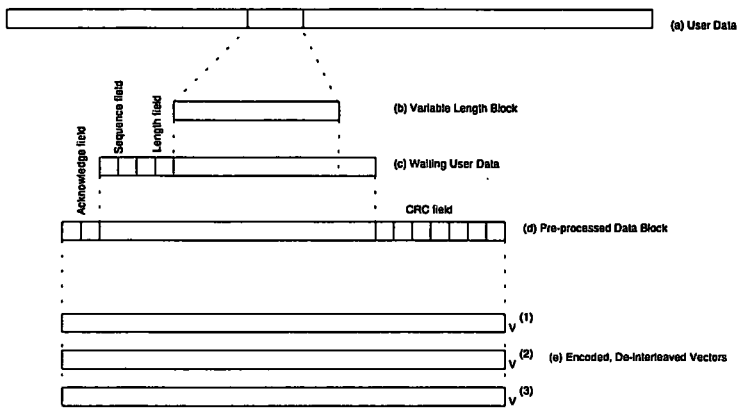


Figure 6-2 **Data Block Formats.**

The *User Data* (a) is the actual data that is to be sent from one user to another. This data is presented to the encoding scheme via a buffer so that the scheme may extract the data in blocks as required. The *Variable Length Block* (b) is a block of up to 100 symbols from the user-data buffer. If there are fewer than 100 symbols in the buffer then all of the data is extracted and included in the block. Martins and Alves (1990) have shown that the best block size to use is dependent on the actual channel error rate. To make use of this benefit it would be necessary to develop a scheme that deals with variable size blocks as well as variable redundancy. As will be

seen, the protocol for the proposed scheme is already complex and it is questionable whether a variable length block scheme with its associated protocol overhead would result in a net increase in throughput.

The *Waiting User Data* (c) has a fixed length and a fixed format. Du, Kasahara & Namekawa (1988) developed a variable rate scheme based on BCH codes with an emphasis on keeping the block length the same by filling unused places with additional data symbols. However, the resulting scheme ended up having a variable format which resulted in a complex protocol which would require a large protocol overhead. Here, effort has been taken to use a fixed length and fixed format to simplify as much as possible the control protocol with an associated reduced protocol overhead.

The data field is 100 symbols long and the length field indicates how much of this is valid user data. It is necessary to have a fixed length data field because the length information will not be available at the receive site until after the decode process is complete. The length field is 6 symbols long and the length value, 0-100, is copied to each group of 3 symbols so that it is duplicated (3 symbols may store values in the range 0 to $6^3-1 = 215$). The sequence field is also 6 symbols long and each group of 3 contains a value, 0-215, that indicates the data fields position in the overall

message. Once a value of 215 is reached, the number begins again from 0.

The *Pre-processed data block* (d) is identical to the *Waiting User Data* (c) except that it has 6 symbols of acknowledge information (more on this later) appended to the start, 2 groups of 3 symbols duplicate the acknowledge number, and seven symbols of CRC information appended to the end. The *Pre-processed data block* is passed to the convolutional encoder which produces the *Encoded de-interleaved vectors* (e).

The acknowledge field is used to tell the other end how much data has been successfully received from it by the local receive scheme. The local receive scheme uses the sequence field in decoded blocks to sequence (i.e. order) successfully received blocks of data. It informs the remote transmit scheme that it has received up to and including block N by sending N,N in the acknowledge field (it may also have successfully received blocks above N but these will not be acknowledged until there is a contiguous group). The CRC field is a 7 symbol check word calculated from the whole of the *Pre-processed data block* excluding the CRC field itself.

6.4 Consideration Of Possible Faults.

Although all data transmitted across the HF channel is protected by a CRC check-word it is possible that errors may go undetected. The scheme discussed above protects 118 symbols of system data (which includes User Data, length, sequence and acknowledge fields) with a 7-symbol CRC check-word. The CRC check-word is computed using an alphabet of 7 characters and produces 6 symbols. These 6 symbols are translated to a base 6 alphabet producing the 7 symbols to be transmitted. The chance of a random error going undetected is 1 in 7^6 = 117,649 and the CRC is able to detect any sequence of 6 or fewer consecutive errors within the message. Even though this error probability is small an error will eventually get through and when it does we must make sure that our scheme will not lock-up.

6.4.1 Error Types.

6.4.1.1 Errors In The User Data Field.

When errors occur in the User Data field they will not cause a system fault. If the user wishes to have a lower undetected-message-error probability than that provided by the underlying communications system then additional error protection must be provided by the user at the application layer.

6.4.1.2 Errors In The Length Field.

When errors occur in the length field it is possible that the User Data field will be either truncated or zero-extended. As stated above, the length of the User Data field is up to 100 symbols and the actual length is stored in duplicate in the length field. For errors in the length field to have any effect both copies of the length value must be corrupted to the same value and these values must be in the range 0-100. Errors of this type will not cause the system to lock-up but the user's message will be corrupted. If the user wishes to have a lower undetected-message-error probability than that provided by the underlying communications system then additional error protection must be provided by the user at the application layer.

6.4.1.3 Errors In The Acknowledge Field.

When errors occur in the acknowledge field it is possible for the system to lock-up if a mechanism to detect this type of error is not provided. For errors in the acknowledge field to have any effect both copies of the acknowledge value must be corrupted to the same value and this value must lie within a narrow range. The range depends on the 'window size' W being used which defines the number of unacknowledged blocks that can be outstanding. As an example, using a window size of 8, consider the following situation:

- all blocks up to and including block 145 have been received.
- the value 145 is sent in the acknowledge field to the far end and received correctly.

During the next transmission the far end will transmit blocks 146 to 154 inclusive and will be expecting an acknowledgement somewhere in the range 145 to 154 (145 if none of the blocks got through and 154 if all got through). If the local decode scheme manages to decode up to and including block 149 then the scheme will send the value 149 in its acknowledge field back to the far end. A problem occurs when this field is corrupted so that both copies are changed to some other value in the range 145 to 154. In the situation where it is changed to a value less than 149 then all that happens is that a few extra blocks are transmitted unnecessarily - the system does not hang and re-synchronises after a maximum of 3 re-transmissions (this is due to the fact that after level 3 decoding has been attempted and failed the system falls-back to level 1 until the channel recovers sufficiently to support error-free transmissions). However, if the field is corrupted to a value in the range 150-154 (i.e. a valid value higher than 149), e.g. 151, then the far end will assume that all blocks up to and including block 151 have been received correctly. The consequence of this is that from now on the remote

transmit site will only ever send blocks with sequence numbers greater than 151 resulting in blocks 150 and 151 never getting acknowledged and the system locking up. Fortunately there is a way of detecting that this has happened. When the channel recovers sufficiently to support error free transmissions the far end will be able to detect that it is receiving the value 149 in the acknowledge field repeatedly and not a value in the range 152 to 160. This is an indication that things have gone wrong and that remedial action is necessary. The remedial action will be for the far end to adjust its sequencing numbers so that instead of sending the next 8 blocks with sequence numbers 152 to 160 it adjusts the sequencing so that they are sent with 150-158 (subsequent blocks will have sequence numbers following on from 158). This will prevent the system from locking-up but will result in the user's message being corrupted. If the user wishes to have a lower undetected-message-error probability than that provided by the underlying communications system then additional error protection must be provided by the user at the application layer.

6.4.1.4 Errors In The Sequence Field.

When errors occur in the sequence field it is possible that the Users Data field will be placed in the wrong position in the user's message (possibly overwriting valid data). For errors in the sequence field to have any effect both copies of the sequence value must be

corrupted to the same value and lie within a specified range. The range depends on the window size being used and can extend from the last block accepted, call it N, to N+(window size). Errors of this type will not cause the system to lock-up but the user's message will be corrupted. If the user wishes to have a lower undetected-message-error probability than that provided by the underlying communications system then additional error protection must be provided by the user at the application layer.

The scheme described above provides a robust and flexible protocol that can be used for any variable rate code that is able to split its various rates into blocks of fixed size. A full description of the coding required to implement this scheme is given in appendix D.

6.5 Chapter Summary.

This chapter has presented a protocol to support the variable rate scheme described in chapter 3. A simple protocol was described first which is adequate if the control-data flowing between transmit and receive sites could be guaranteed error free. The types of error that can occur in 'real-life' were examined next and the consequence of these errors on the protocol determined. It was found possible to develop strategies to deal with all these types of error so that the communications link

could be maintained. In the worst cases, parts of the user's data would be corrupted. However, if the user wishes a higher degree of confidence in his data than that provided by the underlying communications system then he must turn to the application layer to provide this.

7. Conclusions And Areas Of Further Work.

7.1 Conclusions.

It has been shown that the performance of multi-level schemes is much better than the performance of simple binary schemes. The first stage of improvement comes from the equipment demodulators as these are able to spend longer periods of time examining the signal elements. This is because the information in a symbol increases as the choice from which the signal is drawn increases. Hence, for a given information rate the symbol rate may be reduced for a modulation scheme using a larger number of symbols. The second stage of improvement comes from the error-correction scheme. The improved performance of multi-level schemes, over and above simple binary schemes, was demonstrated in chapter 4 section 4.3. It was seen that the block-decoding error probability could be reduced by upto 68% in moving from a simple binary scheme to a scheme employing six symbols. It was also noted that any further increase in the number of symbols used was unlikely to improve the performance significantly.

Chapter 2 discusses the choice of 6-tone MFSK for the new HF equipment being developed and it is shown to be a suitable modulation scheme. With this decision already made the task of the error-correcting system is to provide a robust and reliable scheme that will work well

with the modulation scheme and match the error statistics of the HF channel. Although the chosen modulation scheme does limit the choice of error-correcting schemes it has been shown that a good error-correcting system can still be developed.

The algebra for binary and non binary block codes requires there to be q symbols in the symbol set for the algebra to work where q is either a prime or a power of a prime. The modulation scheme for the new equipment uses 6 symbols which does not satisfy the requirements for q and therefore an error-correcting scheme that does not rely on this theory must be sought. Convolutional codes together with either Viterbi decoding or Fano decoding satisfy this requirement with the advantage that the codes can be made as powerful as required by varying their constraint length.

Having made the decision to use convolutional codes a method used to decode them must be chosen. Two straightforward decoding techniques are Viterbi maximum likelihood decoding and Fano sequential decoding. Appendices A and B describe these techniques and it is seen in chapter 4, where the two are compared, that the performance of the Viterbi decoder is superior to that of the Fano decoder. However, to obtain good decoding performance from a convolutional code it is necessary to use a code with a reasonably long constraint length. As

the constraint length increases the resources required by the Viterbi decoder and the decoding time taken increase exponentially. It is this exponential increase that motivates the search for an alternative decoding strategy. The Fano sequential decoder, whilst sub-optimum, provides a reasonably good alternative to the Viterbi decoder. It is shown in chapter 4 that, for blocks of length 100, the block-decoding error probability falls by around 20% when decoding is switched from Viterbi decoding to Fano decoding. To compensate for this reduction in performance it is possible to increase the memory of the code until the performance of the Fano decoder is increased to the required level. It is noted that this is comparing unlike codes but the point is that the resources required by the Fano decoder are not significantly increased by this change.

Using any coding scheme over a time varying channel places additional requirements on the system. For the system to deliver optimum throughput it must be able to vary its error-correcting power in sympathy with the variation in channel error-rate. It is quite simple to adapt convolutional codes to variable rate operation as was seen in chapter 3. In addition to variable rate being easily achievable it is also easy to make the variable rate code a systematic code so that any transmitted data may be quickly examined to see if it has been received error free. An added advantage of

systematic convolutional codes is that they are always non-catastrophic.

Having developed the variable rate systematic convolutional coding scheme, and a simple protocol to support it, the throughput of the scheme was examined in chapter 5. It was seen that the throughput of the variable rate scheme closely tracked the upper envelope of the throughput for fixed rate schemes. At around 6% channel error-rate the variable rate scheme delivered better throughput than the upper envelope of the fixed rate schemes which is an advantage to the HF system. It is an advantage because, at the highest transmission rates, it was found that the average channel error probability was approximately 6%. The fact that the average channel error probability was at 6% at the highest transmission rate is significant. When the throughput analysis was carried out at various transmission rates it was found that throughput was highest at the highest transmission rates. As commented in chapter 5, this result is not as obvious as it may seem; it was anticipated that the increase in channel errors at the higher transmission rates would result in a net reduction in throughput.

One further enhancement to the coding scheme is the introduction of interleaving. Interleaving is a technique commonly used to break up bursts of errors to

make the channel appear more uniform in its distribution of errors. Interleaving is inherent in the variable rate scheme because the different encoder vectors are sent in different time frames and then interleaved at the receive site prior to decoding. However, channel interleaving was tested on the scheme at the various transmission rates to see if any extra benefit could be gained. It was found that at the highest transmission rate the throughput of the variable rate scheme increased from 329.70 bits/sec to 362.34 bits/sec, an increase of almost 10%.

To complete the work a suitable protocol that would allow the variable rate scheme to be used over a practical HF channel was developed. Chapter 6 discusses the problems with practical channels and details the problems encountered when both the forward and return channels are subject to errors. With these potential problems in mind a protocol that could cope with all foreseeable scenarios was developed. The system was kept as simple as possible and a fixed-length fixed-format block transfer scheme used. The fixed-format was chosen in preference to more elaborate schemes to keep the protocol overhead to a minimum. This is particularly significant when the results of chapter 5, section 5.4.1 are examined. In this section it is seen that a large number of the data transfers are error free which is a major contributing factor to the throughput efficiency of the scheme. If a

complex protocol had been developed the associated protocol overhead would have diminished this advantage.

Full details of the implementation of this scheme are given in appendix D.

7.2 Further Work.

7.2.1 Optimum Block Size.

Throughout this thesis, with the exception of chapter 6, a block size of 100 symbols has been used. The block is built up from 95 user data symbols and 5 CRC symbols. The initial choice of 5 CRC symbols was based on a quick approximation on how many CRC symbols would be needed. The error statistics examined in chapter 5 section 5.4.1 are based on this fact. Obviously the throughput is strongly influenced by the number of blocks that are received error free and by varying the block size there is bound to be a change in the distribution of errors. It would be worth exploring different block sizes to determine which block size gives the best throughput and noting how sensitive the optimum block size is to variation in error distribution.

7.2.2 Introduce Redundancy In Finer Increments.

It would be interesting to determine what happens to the throughput if the variable rate scheme introduced its redundancy in finer increments. It is suspected that

there would be little gained because of the nature of the HF channel. Errors tend to occur in bursts on the HF channel with the consequence that error percentages found in successive blocks change by significant amounts. The effect of this is that the error-correcting scheme needs to be able to change its error-correcting power by similar significant amounts. The use of say puncturing to provide finer increments in redundancy would force the system to step through a number of iterations before the correct level of decoding power was reached. This could result in a reduced throughput simply due to the necessary increase in protocol overhead. An alternative method would be to use some sort of error counting scheme (by sending known bits) to determine the level of redundancy required. Again this would reduce throughput, particularly in the case of error free transmissions. A scheme that used rates 1/1, 3/4, 2/3, 1/2, 1/3 and 1/4 based on a basic 1/4 rate encoder may yield an improvement and could be worth trying.

7.2.3 Codes With Larger Memory Order.

Throughout this thesis the memory of the various convolutional codes has been limited to between 2 and 6. This limitation has been imposed by the analysis tools available particularly where the Viterbi decoder is concerned. To fully explore the potential of the scheme a much larger code memory should be used on a machine that has significantly more processing power and more

memory. With the recent introduction of processors such as the 200MHz Intel Pentium and memory at around £3.00 per Megabyte it is now possible to explore these avenues without the need to use systems found only in large educational institutions and multi-national companies. Lists of optimum distance profile codes have been constructed (Johannesson 1975), (Jonannesson 1977), (Jonannesson & Paaske 1978). These lists detail both systematic and non-systematic codes with memory order upto 35. Use of some of these codes in the scheme developed should give excellent performance and it would be useful to document the performance level achieved.

7.2.4 Maximum Transmission Rate.

It was stated in chapter 5 that the maximum throughput was achieved at the highest data rate. The highest data rate in this case is 600 bits/sec which corresponds to approximately 240 symbols per second (using 5 bits ≈ 2 symbols). It was anticipated that the increase in errors at this higher data rate would result in a net reduction in throughput because of the many retransmissions required. Obviously this was not the case and the apparent near-linear increase in throughput right upto the rate of 600 bits/sec suggests that a further improvement is likely at even higher data rates. Again, to explore the full capabilities of the scheme, it would

be interesting to push the data rate even higher until a maximum is found.

8. References & Acknowledgements.

8.1 References.

Anderson J. B. 1989. Limited search trellis decoding of convolutional codes. *IEEE trans. inform. theory*, vol. 35, no. 5, pp. 944-955.

Berlekamp E. R. 1968. Algebraic coding theory. McGraw-Hill, New York.

Cain J. B., Clark G. C. & Geist J. M. 1979. Punctured convolutional codes of rate (n-1)/n and simplified maximum likelihood decoding. *IEEE trans. inform. theory*, vol. 25, no. 1, pp. 97-100.

Calderbank A. R., Mazo J. E. & Wei V. K. 1985. Asymptotic upper bounds on the minimum distance of trellis codes. *IEEE trans. comm.*, vol. 33, no. 4, pp. 305-309.

Cedervall M. & Johannesson R. 1989. A fast algorithm for computing distance spectrum of convolutional codes. *IEEE trans. inform. theory*, vol. 35, no. 6, pp. 1146-1159.

Chase D. 1985. Code combining - a maximum-likelihood decoding approach for combining an arbitary number of noisy packets. *IEEE trans. comm.*, vol. 33, no. 5, pp. 385-393.

Chevillat P. R. & Costello D. J. Jr. 1976. Distance and computation in sequential decoding. *IEEE trans. comm.*, vol. 24, pp. 440-447.

Collins O. M. & Hizlan M. 1993. Determinate state convolutional codes. *IEEE trans. comm.*, vol. 41, no. 12, pp. 1785-1794.

Du J., Kasahara M. & Namekawa T. 1988. Separable codes on type-II hybrid ARQ systems. *IEEE trans. comm.*, vol. 36, no. 10, pp. 1089-1097.

Elias P. 1955. Coding for noisy channels. *IRE Conv. Rec.*, vol. 3, part4, pp. 37-46.

Fano R. M. 1963. A heuristic discussion of probabilistic decoding. *IEEE trans. inform. theory*, vol. IT-9, pp. 64-74.

Fantacci R. 1990. Performance evaluation of efficient continious ARQ protocols. *IEEE trans. comm.*, vol. 38, no. 6, pp. 773-781.

Fettweis G. & Meyr H. 1989. Parallel Viterbi algorithm implementation: breaking the ACS-bottleneck. *IEEE trans. comm.*, vol. 37, no. 8, pp. 785-789.

Gallager R. G. 1965. A simple derivation of the coding theorem and some applications. *IEEE trans. inform. theory*, vol. 11, pp. 3-18.

Goodman R. M. F. & Farrell P. G. 1975. Data transmission with variable-redundancy error control over a high-frequency channel. *Proc. IEE*, vol. 122, no. 2, pp. 113-118.

Hagelbarger D. W. 1959. Recurrent codes: easily mechanised, burst-correcting, binary codes. *Bell sys. tech. j.*, vol. 38, pp. 969-984.

Hagenauer J. 1988. Rate-compatible punctured convolutional codes (RCPC codes) and their applications. *IEEE trans. comm.*, vol. 36, no. 4, pp. 389-400.

Hegde M. V., Naraghi-Pour M. & Chen X. 1994. Convolutional coding for finite-state channels. *IEEE trans. comm.*, vol. 42, no. 6, pp. 2231-2238.

Jeffrey A. 1979. Mathematics for engineers and scientists, second edition. Van Nostrand Reinhold (UK), pp. 383-386.

Jelinek F. 1969. Fast sequential decoding algorithm using a stack. *IBM j. res. and dev.*, vol. 13, pp. 675-678.

Johannesson R. 1975. Robustly-optimal rate one-half binary convolutional codes. *IEEE trans. inform. theory*, vol. 21, pp. 464-468.

Jonannesson R. 1977. Some rate 1/3 and 1/4 binary convolutional codes with an optimum distance profile. *IEEE trans. inform. theory*, vol. 23, pp. 281-283.

Jonannesson R. & Paaske E. 1978. Further results on binary convolutional codes with an optimum distance profile *IEEE trans. inform. theory*, vol. 24, pp. 264-268.

Kallel S. 1988. Sequential decoding with ARQ and code combining: a robust hybrid FEC/ARQ system. *IEEE trans. comm.*, vol. 36, no. 7, pp. 773-780.

Kallel S. 1990. Analysis of a type II hybrid ARQ scheme with code combining. *IEEE trans. comm.*, vol. 38, no. 8, pp. 1133-1137.

Kallel S. & Haccoun D. 1990. Generalized type II hybrid ARQ scheme using punctured convolutional coding. *IEEE trans. comm.*, vol. 38, no. 11, pp. 1938-1946.

Kallel S. & Haccoun D. 1991. Sequential decoding with an efficient partial retransmission ARQ strategy. *IEEE trans. comm.*, vol. 39, no. 2, pp. 208-213.

Kallel S. 1992. Sequential decoding with an efficient incremental redundancy ARQ scheme. *IEEE trans. comm.*, vol. 40, no. 10, pp. 1588-1593.

Krishna H. & Morgera S. D. 1987. A new error control scheme for hybrid ARQ systems. *IEEE trans. comm.*, vol. 35, no. 10, pp. 981-990.

Lee L. H. C. 1994. New rate-compatible punctured convolutional codes for Viterbi decoding. *IEEE trans. comm.*, vol. 42, no. 12, pp. 3073-3079.

Lee T. H. 1991. Performance analysis of efficient concatenated coding scheme for ARQ protocols. *IEE Proc I*, vol. 138, no. 3, pp. 148-152.

Leonard D. A. & Rodger C. A. 1989. Limiting error propagation in Viterbi decoding of convolutional codes. *IEEE trans. inform. theory*, vol. 35, no. 6, pp. 1295-1299.

Lin S. & Costello J. D. Jr. 1983. Error control coding: fundamentals and applications. Prentice-Hall, Englewood Cliffs, NJ.

Lugand L. R. & Costello D. J. Jnr. 1982. A comparison of three hybrid ARQ schemes using convolutional codes on a nonstationary channel. *Conf. Rec., GLOBECOM '82, Miami, FL, Nov. 29 - Dec. 2, 1982*, pp. C8.4.1 - C8.4.5.

Ma H. H. & Wolf J. K. 1986. On tail biting convolutional codes. *IEEE trans. comm.*, vol. 34, no. 2, pp. 104-111.

Mandelbaum D. M. 1974. Adaptive-feedback coding scheme using incremental redundancy. *IEEE trans. inform. theory*, vol 20, pp. 388-389.

Martins J. A. C. & Alves J. D. C. 1990. ARQ protocols with adaptive block size perform better over a wide range of bit error rates. *IEEE trans. comm.*, vol. 38, no. 6, pp. 737-739.

Mason S. & Zimmerman H. 1960. Electronic Circuits, Signals, and Systems. Wiley, New York.

Michelson A. M. & Levesque A. H. 1985. Error control techniques for digital communication. Wiley-Interscience, New York.

Morgera S. D. & Oduol V. K. 1989. Soft-decission decoding applied to the generalised type-II hybrid ARQ scheme. *IEEE trans. comm.*, vol. 37, no. 4, pp. 393-396.

Peterson W. W. 1960. Encoding and error-correction procedures for the Bose-Chaudhuri codes. *IRE trans. inform. theory*, vol. IT-6, pp. 459-470.

Peterson W. W. 1961. Error correcting codes. MIT Press, Cambridge, MA.

Ping S., Yan Y. & Feng C. 1991. An effective simplifying scheme for Viterbi decoder. *IEEE trans. comm.*, vol. 39, no. 1, p. 1-3.

Prange E. 1957. Cyclic error-correcting codes in two symbols. *Tech. Note AFCRC-TN-57-103*, Air Force Cambridge Research Center, Cambridge, Mass.

Ralphs J. D. 1985. Principles & practice of multi-frequency telegraphy. IEE Telecommunications Series; II, Peter Peregrinus.

Reed I. S. & Solomon G. 1960. Polynomial codes over certain finite fields. *J. soc. ind. appl. math.*, vol. 8, pp. 300-304.

Savage J. E. 1966. The distribution of sequential decoding time. *IEEE trans. inform. theory*, vol. IT-12, pp. 143-147.

Schlegel C. B. & Herro M. A. 1990. A burst-error-correcting Viterbi algorithm. *IEEE trans. comm.*, vol. 38, no. 3, pp. 285-291.

Shannon C. E. 1948. A mathematical theory of communication. *Bell sys. tech. j.* no. 27, pp. 379-423, pp. 623-656.

Viterbi A. J. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE trans. inform. theory*, vol. IT-13, no. 2, pp. 260-269.

Vucetic B. 1991. An adaptive coding scheme for time-varying channels. *IEEE trans. comm.*, vol. 39, no. 5, pp. 653-663.

Wang Q. & Bhargava V. K. 1989. An efficient maximum likelihood decoding algorithm for generalised tail biting convolutional codes including quasicyclic codes. *IEEE trans. comm.*, vol. 37, no. 8, pp875-879.

Wang Q., Li G., Bhargava V. K. & Mason L. J. 1993. Repeated convolutional codes for high-error-rate

channels. *IEEE trans. comm.*, vol. 41, no. 6, pp. 852-863.

Wang Y. & Lin S. 1983. A modified selective-repeat type-II hybrid ARQ system and its performance analysis. *IEEE trans. comm.*, vol. 31, no. 5, pp. 593-608.

Wen K., Wen T. & Wang J. 1990. A new transform algorithm for Viterbi decoding. *IEEE trans. comm.*, vol. 38, no. 6, pp. 764-772.

Wicker S. B. & Bartz M. J. 1994. Type-II hybrid-ARQ protocols using punctured MDS codes. *IEEE trans. comm.*, vol. 42, no. 2/3/4, pp. 1431-1440.

Wozencraft J. M. & Reiffen B. 1961. Sequential decoding. MIT Press, Cambridge, Mass., pp. 25-46.

Yasuda Y., Kashiki K. & Hirata Y. 1984. High-rate punctured convolutional codes for soft decission Viterbi decoding. *IEEE trans. comm.*, vol. 32, no. 3, pp. 315-319.

Zigangirov K. 1966. Some sequential decoding procedures. *Problems of Information Transmission* (In English), vol. 2, no. 4, pp. 1-10.

8.2 Acknowledgements.

The author would like to express his sincere thanks to Dr. Mike Meade for his friendship, enthusiasm and support throughout the period of research and study. Thanks also go to Proffessor Farrel and Dr. Bissel for their comments regarding the presentation of this final version of the thesis, to my wife Jessica for her tolerance and loving support and to my family for their encouragement over the years.

A.1. Convolutional Codes.

A.1.1 Introduction.

The main sections of this thesis are concerned with the performance of multi-level convolutional codes. For this reason a detailed look at their structure is presented here.

Convolutional codes were first studied by Elias (1955) and introduced by him. Viterbi (1967) showed that convolutional codes have the ability to perform better than block codes. He showed that in the extremes (Rate = Channel Capacity and Rate = 0) Convolutional codes and block codes are approximately equal, but for intermediate rates convolutional codes perform better than block codes. Convolutional codes therefore represent a good alternative to block codes and shortly after their introduction a number of efficient decoding algorithms started to appear. This appendix will examine the structure and encoding of convolutional codes. A discussion of decoding methods is covered in appendix B and appendix C.

A.1.2 Encoding Of Convolutional Codes.

Convolutional encoders form n output symbols based not only on their present k input symbols, but also on their m previous inputs. In this way the past history of the information symbols input to the encoder is apparent in

the output sequence. The extent to which this happens depends on the memory m of the encoder (the number of old inputs still in the encoder) and affects the performance of the code.

The *constraint length* of a convolutional encoder is defined as:

$$n_A = n(m+1)$$

This relationship stems from the fact that an information symbol remains in the encoder for upto $m+1$ time units and can affect any of the n encoder outputs during each time unit. n_A can be interpreted as the maximum number of encoder output symbols that can be affected by a single information symbol.

A convolutional encoder is a linear system. As such its output sequence can be obtained by convolving its input with the encoder 'impulse response' (i.e. the outputs obtained when the sequence 100000000 . . . . . is applied to the input).

Consider the simple encoder shown in figure A-1.

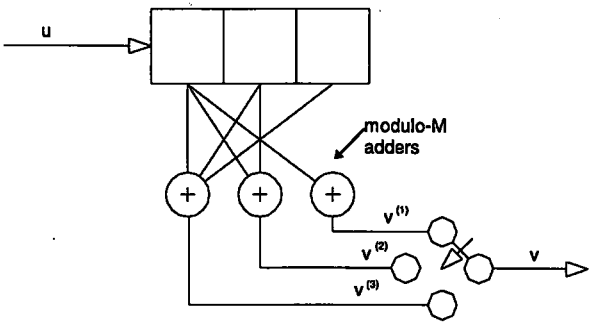


***Figure A-1* A simple convolutional encoder.**

In this encoder m=2 and there are k=1 inputs per unit time with n=3 outputs. Applying the impulse $u=\{1\,0\,0\,0\,0\,0\,..\,.0\}$ at the encoder input it may be verified that the encoder outputs will be $v^{(1)}=\{1\,0\,0\,0\,0\,0\,...0\}$, $v^{(2)}=\{1\,1\,0\,0\,0\,0\,...0\}$, $v^{(3)}=\{1\,1\,1\,0\,0\,0\,...0\}$. Since the encoder has a memory of size m, the impulse responses can last at most m+1 time units and the output vectors can be truncated to length m+1 = 3. These truncated vectors are called the generator sequences: $g^{(1)}$, $g^{(2)}$ and $g^{(3)}$. Thus the generator sequences for the encoder in figure A-1 are:

$$g^{(1)}=\{g_0^{(1)}\,g_1^{(1)}\,g_2^{(1)}\}=\{1\,0\,0\};$$

$$g^{(2)}=\{g_0^{(2)}\,g_1^{(2)}\,g_2^{(2)}\}=\{1\,1\,0\};$$

$$g^{(3)}=\{g_0^{(3)}\,g_1^{(3)}\,g_2^{(3)}\}=\{1\,1\,1\}.$$

Using the generator sequences, the outputs $v^{(1)}$, $v^{(2)}$ and $v^{(3)}$ can be obtained from the equations:

$$v^{(1)} = u * g^{(1)}$$

$$v^{(2)} = u * g^{(2)}$$

$$v^{(3)} = u * g^{(3)}$$

where * denotes discrete time convolution.

If the generator sequences are interlaced and arranged in a matrix:

$$G = \begin{bmatrix} g_0^{(1)} g_0^{(2)} g_0^{(3)} & g_1^{(1)} g_1^{(2)} g_1^{(3)} & \cdots g_m^{(1)} g_m^{(2)} g_m^{(3)} & & & & & & & & & & & & & & \\ \cdot & \cdot & \cdot & g_0^{(1)} g_0^{(2)} g_0^{(3)} & g_1^{(1)} g_1^{(2)} g_1^{(3)} & \cdots g_m^{(1)} g_m^{(2)} g_m^{(3)} & & & & & & & & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & g_0^{(1)} g_0^{(2)} g_0^{(3)} & g_1^{(1)} g_1^{(2)} g_1^{(3)} & \cdots g_m^{(1)} g_m^{(2)} g_m^{(3)} & & & & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & & & & & & & & & & & & & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

where all the blank areas are filled with zeros then the output can be expressed as:

$$v = uG$$

G is called the *generator matrix* of the code.

To clarify the procedure consider the input of the sequence $u = (1\,0\,0\,1\,0\,1)$ to the simple encoder shown in figure A-1. All operations are carried out using modulo-2 arithmetic (since this simple encoder is a binary one), and because the input sequence is of finite length (6 time units) then the generator matrix is of finite size

and the output sequence is of finite length. The encoding equations become:

$$v = uG$$

$$v = \{100101\}\begin{bmatrix} 111 & 011 & 001 & 000 & 000 & 000 & 000 & 000 \\ 000 & 111 & 011 & 001 & 000 & 000 & 000 & 000 \\ 000 & 000 & 111 & 011 & 001 & 000 & 000 & 000 \\ 000 & 000 & 000 & 111 & 011 & 001 & 000 & 000 \\ 000 & 000 & 000 & 000 & 111 & 011 & 001 & 000 \\ 000 & 000 & 000 & 000 & 000 & 111 & 011 & 001 \end{bmatrix}$$

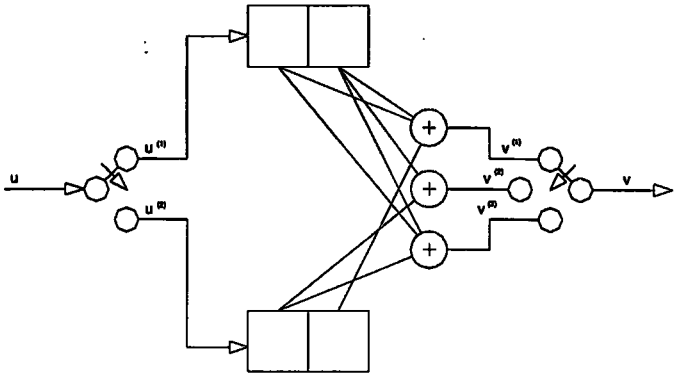
$$\{111\ 011\ 001\ 111\ 011\ 110\ 011\ 001\}$$

As a more complex example, consider the encoder in figure A-2a which can be re-drawn as in figure A-2b.

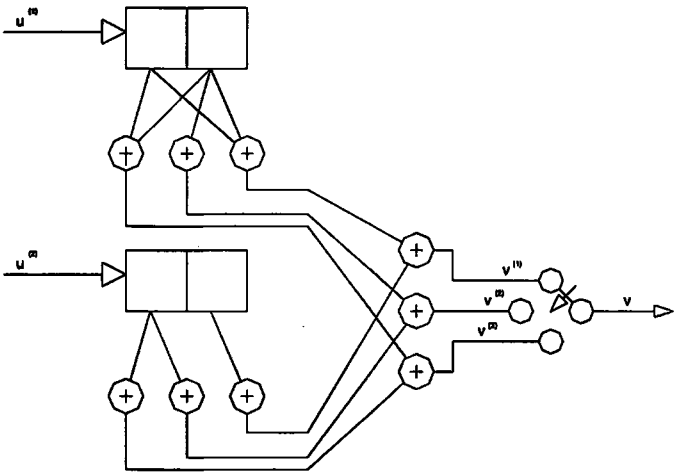
The input sequence now consists of 2 symbols per time unit and can be expressed as $u = \{u_0^{(1)}u_0^{(2)}, u_1^{(1)}u_1^{(2)}, u_2^{(1)}u_2^{(2)}, \ldots\}$ or as two separate inputs $u^{(1)}=\{u_0^{(1)}, u_1^{(1)}, u_2^{(1)}, \ldots\}$ and $u^{(2)}=\{u_0^{(2)}, u_1^{(2)}, u_2^{(2)}, \ldots\}$. Consequently there are now two sets of generator sequences, one for each input $u^{(i)}$, and these are obtained by determining the impulse response of each half of the encoder:

$$g_1^{(1)} = 11,\ g_1^{(2)} = 01,\ g_1^{(3)} = 11$$

$$g_2^{(1)} = 01,\ g_2^{(2)} = 10,\ g_2^{(3)} = 10$$



(a) Encoder drawn conventionally.



(b) Encoder drawn to reveal its structure.

***Figure A-2* A more complex convolutional encoder.**

To form the generator matrix for this encoder, the g_1s are interleaved to give 101 111 and the g_2s interleaved to give 011 100. Now, the overall output of the encoder is obtained by linearly combining the outputs of each half encoder, as shown in figure A-2b. For this reason the second interleaved sequence is placed below the first

in the generator matrix. The resulting generator matrix is as shown below:

$$G=\begin{bmatrix} 101 & 111 & 000 & 000 & . & . \\ 011 & 100 & 000 & 000 & . & . \\ 000 & 101 & 111 & 000 & . & . \\ 000 & 011 & 100 & 000 & . & . \\ 000 & 000 & 011 & 100 & . & . \\ . & . & . & . & . & . \end{bmatrix}$$

If the input sequence $u = (11, 01, 10)$ is applied to the input of the encoder shown in figure A-2a the output will *be* $v=(110, 000, 001, 111)$ as can be verified.

A convolutional code that has n output symbols for each k input symbols and has a memory of m is described as a (n,k,m) code. In the case of a general (n,k,m) code, the generator matrix is:

$$G=\begin{bmatrix} G_0G_1G_2...G_m & & & \\ & G_0G_1G_2...G_m & & \\ & & G_0G_1G_2...G_m & \\ & & & . \quad . \quad . \quad . \\ & & & . \quad . \quad . \quad . \end{bmatrix}$$

where each G_i is a k x n sub-matrix:

$$G_i=\begin{bmatrix} g_{1,i}^{(1)} g_{1,i}^{(2)} & . & . & . g_{1,i}^{(n)} \\ g_{2,i}^{(1)} g_{2,i}^{(2)} & . & . & . g_{2,i}^{(n)} \\ . & . & . & . \quad . \\ g_{k,i}^{(1)} g_{k,i}^{(2)} & . & . & . g_{k,i}^{(n)} \end{bmatrix}$$

In a (n,k,m) convolutional code there are n output symbols for every k input symbols and therefore the code rate is $R = k/n$. Treating the convolutional encoder as a special case of a block code and applying an input sequence of $kL+m$ symbols (the final m input symbols being all zeros to 'flush' the encoder) there will be an output of $n(L+m)$ symbols; this will give rise to a 'block code rate' of $(kL+m)/n(L+m)$ which will be approximately equal to the code rate R if $L >> m$. The amount by which the true block rate $(kL+m)/n(L+m)$ is less than the ideal rate (k/n) is called the rate loss associated with the zero tail. Methods of reducing this rate loss have been explored (Ma & Wolf 1986), (Wang & Bhargava 1989).

The above description of convolutional codes using generator matrices and sub-matrices can become cumbersome especially when $k > 1$. For this reason an alternative description is needed.

In any linear time-invariant system, time-domain operations involving convolution can be replaced by transform-domain operations using multiplication. Each sequence in the encoding equations above can be replaced by a polynomial representation. The sequence of symbols is converted to a polynomial in x by taking the symbols in the sequence as the coefficients in the polynomial. The invariant 'x' in the polynomial

represents a delay factor and the power of x represents the number of time units each symbol has been delayed. To clarify this consider the sequence 1 3 1 2 4 3 1 in polynomial form; this would become $1 + 3x + x^2 + 2x^3 + 4x^4 + 3x^5 + x^6$. As an example of this method consider the simple encoder shown in figure A-1 which is repeated below for convenience.

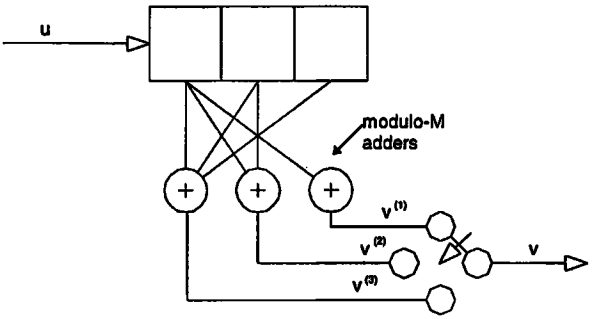


***Figure A-1* A simple convolutional encoder.**

The encoding equations now become:

$$v^{(1)}(x) = u(x)g^{(1)}(x)$$

$$v^{(2)}(x) = u(x)g^{(2)}(x)$$

$$v^{(3)}(x) = u(x)g^{(3)}(x)$$

and

$$v(x) = v^{(1)}(x^3) + xv^{(2)}(x^3) + x^2v^{(3)}(x^3)$$

where

$$g^{(1)} = 1$$

$$g^{(2)} = 1 + x$$

$$g^{(3)} = 1 + x + x^2$$

and

$$u(x) = 1 + x^3 + x^5$$

giving

$$v^{(1)}(x) = 1 + x^3 + x^5$$

$$v^{(2)}(x) = (1 + x^3 + x^5)(1 + x)$$

$$= 1 + x + x^3 + x^4 + x^5 + x^6$$

$$v^{(3)}(x) = (1 + x^3 + x^5)(1 + x + x^2)$$

$$= 1 + x + x^2 + x^3 + x^4 + x^6 + x^7$$

hence

$$v^{(1)}(x^3) = 1 + x^9 + x^{15}$$

$$v^{(2)}(x^3) = 1 + x^3 + x^9 + x^{12} + x^{15} + x^{18}$$

$$v^{(3)}(x^3) = 1 + x^3 + x^6 + x^9 + x^{12} + x^{18} + x^{21}$$

and

$$xv^{(2)}(x^3) = x + x^4 + x^{10} + x^{13} + x^{16} + x^{19}$$

$$x^2v^{(3)}(x^3) = x^2 + x^5 + x^8 + x^{11} + x^{14} + x^{20} + x^{23}$$

so that

$$v(x) = 1 + x^9 + x^{15} + x + x^4 + x^{10} + x^{13} + x^{16} + x^{19} + x^2 + x^5 + x^8 + x^{11} + x^{14} + x^{20} + x^{23}$$

$$= 1 + x + x^2 + x^4 + x^5 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14} + x^{15} + x^{16} + x^{19} + x^{20} + x^{23}$$

i.e.:

v = 1 1 1 0 1 1 0 0 1 1 1 1 0 1 1 1 1 0 0 1 1 0 0 1 - as before.

The generator polynomials can be deduced directly from the encoder circuit diagram. Each $g_k^{(i)}$ is constructed by placing a 1 where a stage in the kth shift register of the encoder is connected to the ith output and a 0 where there is no connection. To see this consider the encoder in figure A-1: $g^{(1)}$ = 1 1 1, corresponding to a generator polynomial of $1 + x + x^2$.

Because the last stage in any $(n,k=1,m)$ code encoder must be connected to at least one output, the degree of at least one generator polynomial must be equal to the shift register memory m.

$$m = \max_{1<j\leq n}\left[deg\ g^{(j)}(x)\right]$$

In a more general (n,k,m) code there are n generator polynomials for each of the k inputs. For the ith shift register, K_i is the number of stages in the register and is given by:

$$K_i = \max_{1<j\leq n}\left[deg\ g^{(j)}(x)\right]$$

and m is determined by the number of delay elements in the longest shift register, i.e.:

$$m = \max_{1<j\leq k} K_i = \max_{\substack{1<j\leq n \\ 1<i\leq k}}\left[deg\ g_i^{(j)}(x)\right]$$

Each $g_i^{(j)}$ can be thought of as relating the ith input polynomial $u^{(i)}(x)$ to the jth output polynomial $v^{(j)}(x)$. Using this, the general k-input, n-output linear system can have its transfer-function represented by a matrix of kxn $g_i^{(j)}$s referred to as a *transfer-function matrix*. Now the encoding equations become:

$$V(x) = U(x)\,G(x)$$

where $U(x) = [u^{(1)}(x), u^{(2)}(x), \ldots u^{(k)}(x)]$ is the k-tuple of input sequences and $V(x) = [v^{(1)}(x), v^{(2)}(x), \ldots v^{(n)}(x)]$ is the n-tuple of output sequences. After multiplexing, the codeword becomes:

$$v(x) = v^{(1)}(x^n) + xv^{(2)}(x^n) + \ldots + x^{n-1}v^{(n)}(x^n).$$

A.1.3 Structural Properties Of Convolutional Codes.

The encoding operations of a convolutional encoder can be represented by a state diagram. The state diagram consists of a number of nodes with branches connecting nodes together. If the encoder consists of k shift registers and the length of the ith shift register is represented by l_i then the total encoder memory l is given by:

$$l = \sum_{i=1}^{k} l_i$$

Each state in the state diagram is denoted by S_i where i corresponds to the l-tuple stored within the encoder. For an encoder supporting an alphabet of q characters there will be q^l states and q^k branches leaving each state (one for each of the possible inputs). Each branch will be labelled with the k-tuple representing the input sequence that caused the transition and the n-tuple representing the output.

Determining the output of a given encoder is simply a matter of tracing a path through the state diagram according to the input sequence applied and making a note of the output sequences on each branch. The operation is complete when the encoder has been returned to the all-zero state by a final sequence of l zeros applied to the input.

As an example consider the binary encoder shown in figure A-1. The state diagram for this encoder is shown in figure A-3

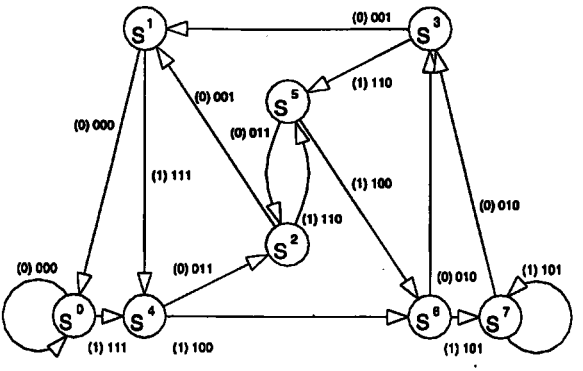


***Figure A-3* A convolutional code state diagram.**

If the input sequence $u=\{1,0,0,1,0,1,0,0\}$ were applied to this encoder (the last two zeros return the encoder to the all-zero state) the output would be $v = \{111, 011, 001, 111, 011, 110, 011, 001\}$ which agrees with the result previously obtained.

The encoder state diagram can be modified to provide a means of determining the weights of all non-zero codewords. The state S^0 is split into an initial state and a final state (the self-loop around state 0 being removed) and branches are labelled with their *'branch gain'* X^i where i represents the weight of the code along that branch. All non-zero codewords are obtained by examining all paths that leave the initial state and terminate at the final state exactly once.

The 'path gain' is the product of the individual branch gains and the weight of the corresponding codeword is the power of X in the path gain.

Considering the modified state diagram as a signal flow chart and applying Mason's gain formula a *generating function* can be obtained (Mason & Zimmerman 1960):

$$T(x) = \sum_i A_i X^i$$

A_i = Number of codewords with weight i.

A path linking the initial state with the final state without passing through any state twice is called a forward path.

Let F_i = gain of ith forward path.

A path starting at any state S_i and returning to it without passing through any other state more than once is called a loop.

Let C_i = gain of ith loop.

A set of loops is non-touching if no state belongs to more than one loop in the set.

Let:

$\{i\}$ = set of all loops;

$\{i',j'\}$ = set of all pairs of non-touching loops;

$\{i'',j'',l''\}$ = set of all triplets of non-touching loops;

and so on.

Now Define:

$$\Delta = 1 - \sum_{i} C_i + \sum_{i'j'} C_{i'} C_{j'} - \sum_{i''j''l''} C_{i''} C_{j''} C_{l''} + \ldots$$

Δ_i is defined as for Δ but excludes the portion of the graph touching the ith forward path (i.e. all states along the ith forward path together with the branches connected to them are removed).

Using Mason's formula we have:

$$T(x) = \frac{\sum_{i} F_i \Delta_i}{\Delta}$$

The modified state diagram used to identify the various paths and loops is shown in figure A-4

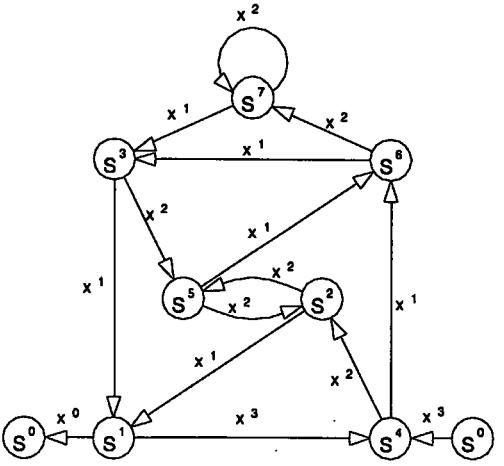


***Figure A-4* Modified state diagram for convolutional code.**

To clarify the procedure, consider the encoder in figure A-1 and the corresponding modified state diagram shown in figure A-4. The state diagram has 7 forward paths:

Fwd Path	States Visited	F_i	
1	$S^0 S^4 S^2 S^5 S^6 S^7 S^3 S^1 S^0$	$F_1=X^{(3+2+2+1+2+1+1+0)}$	$= X^{12}$
2	$S^0 S^4 S^2 S^5 S^6 S^3 S^1 S^0$	$F_2=X^{(3+2+2+1+1+1+0)}$	$= X^{10}$
3	$S^0 S^4 S^2 S^1 S^0$	$F_3=X^{(3+2+1+0)}$	$= X^6$
4	$S^0 S^4 S^6 S^7 S^3 S^5 S^2 S^1 S^0$	$F_4=X^{(3+1+2+1+2+2+1+0)}$	$= X^{12}$
5	$S^0 S^4 S^6 S^7 S^3 S^1 S^0$	$F_5=X^{(3+1+2+1+1+0)}$	$= X^8$
6	$S^0 S^4 S^6 S^3 S^5 S^2 S^1 S^0$	$F_6=X^{(3+1+1+2+2+1+0)}$	$= X^{10}$
7	$S^0 S^4 S^6 S^3 S^1 S^0$	$F_7=X^{(3+1+1+1+0)}$	$= X^6$

There are also 11 loops:

Loop	States visited	C_i	
1	$S^1 S^4 S^6 S^7 S^3 S^5 S^2 S^1$	$C_1 = X^{(3+1+2+1+2+2+1)}$	$= X^{12}$
2	$S^1 S^4 S^6 S^7 S^3 S^1$	$C_2 = X^{(3+1+2+1+1)}$	$= X^8$
3	$S^1 S^4 S^6 S^3 S^5 S^2 S^1$	$C_3 = X^{(3+1+1+2+2+1)}$	$= X^{10}$
4	$S^1 S^4 S^6 S^3 S^1$	$C_4 = X^{(3+1+1+1)}$	$= X^6$
5	$S^1 S^4 S^2 S^5 S^6 S^7 S^3 S^1$	$C_5 = X^{(3+2+2+1+2+1+1)}$	$= X^{12}$
6	$S^1 S^4 S^2 S^5 S^6 S^3 S^1$	$C_6 = X^{(3+2+2+1+1+1)}$	$= X^{10}$
7	$S^1 S^4 S^2 S^1$	$C_7 = X^{(3+2+1)}$	$= X^6$
8	$S^2 S^5 S^2$	$C_8 = X^{(2+2)}$	$= X^4$
9	$S^3 S^5 S^6 S^7 S^3$	$C_9 = X^{(2+1+2+1)}$	$= X^6$
10	$S^3 S^5 S^6 S^3$	$C_{10} = X^{(2+1+1)}$	$= X^4$
11	$S^7 S^7$	$C_{11} = X^{(2)}$	$= X^2$

There are 10 pairs of non-touching loops:

Loops	$C_i C_j$
2 & 8	$C_2 C_8 = X^{8+4} = X^{12}$
3 & 11	$C_3 C_{11} = X^{10+2} = X^{12}$
4 & 8	$C_4 C_8 = X^{6+4} = X^{10}$
4 & 11	$C_4 C_{11} = X^{6+2} = X^8$
6 & 11	$C_6 C_{11} = X^{10+2} = X^{12}$
7 & 9	$C_7 C_9 = X^{6+6} = X^{12}$
7 & 10	$C_7 C_{10} = X^{6+4} = X^{10}$
7 & 11	$C_7 C_{11} = X^{6+2} = X^8$
8 & 11	$C_8 C_{11} = X^{4+2} = X^6$
10 & 11	$C_{10} C_{11} = X^{4+2} = X^6$

Lastly, there are two triples of non-touching loops:

Loops	$C_iC_jC_l$
4, 8 & 11	$C_4C_8C_{11} = X^{6+4+2} = X^{12}$
7, 10 & 11	$C_7C_{10}C_{11} = X^{6+4+2} = X^{12}$

From this information Δ can be determined:

$$\Delta = 1 - (X^{12} + X^8 + X^{10} + X^6 + X^{12} + X^{10} + X^6 + X^4 + X^6 + X^4 + X^2) + (X^{12} + X^{12} + X^{10} + X^8 + X^{12} + X^{12} + X^{10} + X^8 + X^6 + X^6) - (X^{12} + X^{12})$$

$$= 1 - X^2 - 2X^4 - X^6 + X^8$$

As stated above, the Δ_is are determined in exactly the same way as the Δs except that the parts of the graph touching the *i*th forward path are excluded. Examination of figure A-4 and the tables of Forward Paths and Loops reveals that:

i) forward paths 1 and 4 touch every state and therefore there are no loops to be considered when calculating Δ_1 and Δ_4 so that:

$$\Delta_1 = \Delta_4 = 1$$

ii) the part of the graph not touching forward paths 2 and 6 is the loop C_{11} so that:

$$\Delta_2 = \Delta_6 = 1 - X^2$$

iii) the parts of the graph not touching forward path 3 are loops C_{10} and C_{11} giving:

$$\Delta_3 = 1 - (X^2 + X^4) + (X^6)$$

$$= 1 - X^2 - X^4 + X^6$$

iv) the part of the graph not touching forward path 5 is the loop C_8 giving

$$\Delta_5 = 1 - X^4$$

v) finally, the parts of the graph not touching forward path 7 are the loops C_8 and C_{11} giving:

$$\Delta_7 = 1-(X^4 + X^2) + (X^6)$$

$$= 1 - X^2 - X^4 + X^6$$

Combining these results yields:

$$\sum \Delta_i F_i = X^{12} + X^{10}(1 - X^2) + X^6(1 - X^2 - X^4 + X^6) + X^{12} + X^8(1 - X^4) + X^{10}(1 - X^2) +$$

$$X^6(1 - X^2 - X^4 + X^6)$$

$$= 2X^6 - X^8 + X^{12}$$

Therefore:

$$T(X) = (2X^6 - X^8 + X^{12}) / (1 - X^2 - 2X^4 - X^6 + X^8)$$

$$= 2X^6 + X^8 + 5X^{10} + 9X^{12} + 18X^{14} + \ldots\ldots$$

Hence there are 2 paths with weight 6, 1 path with weight 8, 5 paths with weight 10, 9 paths with weight 12, 18 paths with weight 14 and so on.

If the modified state diagram is now further modified so that each branch is also labelled with Y^j and Z ,where j

equals the weight of the k information digits, then the generating function becomes:

$$T(X,Y,Z)=\sum_{i,j,l} A_{i,j,l} X^i Y^j Z^l$$

where:

$A_{i,j,l}$ = number of paths with codewords of weight i, information sequences of weight j and traversing l branches.

and the generating function now gives a complete description of the code. Wicker and Bartz (1994) have shown that a code's signal flow graph and generating function can also be used to assess its throughput and reliability.

A.1.4 Systematic Convolutional Codes.

A special subclass of convolutional codes are <u>systematic codes</u>. In systematic codes the first k outputs from the encoder correspond to the k-digit information sequence applied to the input of the encoder:

$$v_i = u_i \qquad i = 1,2,\ldots,k$$

$$g_i^{(j)} = 1 \text{ if } j = i, \ = 0 \text{ otherwise} \qquad i = 1,2,\ldots,k$$

$$G=\begin{bmatrix} I & P_0 & 0 & P_1 & 0 & P_2 & \cdots & 0 & P_m & & \\ & I & P_0 & 0 & P_1 & \cdots & 0 & P_{m-1} & P_m & \\ & & I & P_0 & \cdots & 0 & P_{m-2} & P_{m-1} & P_m \end{bmatrix}$$

where:

I is the k x k identity matrix,

0 is the k x k all-zero matrix and

$\mathbf{P}_i$ is the k x $(n-k)$ matrix:

$$P_i = \begin{bmatrix} g_{1,i}^{(k+1)} & g_{1,i}^{(k+2)} & \cdots & g_{1,i}^{(k+n)} \\ g_{1,i}^{(k+1)} & g_{1,i}^{(k+2)} & \cdots & g_{1,i}^{(k+n)} \\ \cdot & \cdot & \cdots & \cdot \\ g_{1,i}^{(k+1)} & g_{1,i}^{(k+2)} & \cdots & g_{1,i}^{(k+n)} \end{bmatrix}$$

Systematic codes have a number of advantages over non-systematic codes:

i) Only $k(n-k)$ sequences must be specified to define a systematic code.

ii) Systematic encoders can be realised using less hardware.

iii) No inverting circuit is needed.

iv) They are always *noncatastrophic* (a *catastrophic code* is one in which an infinite number of decoding errors can occur as a result of a finite number of digit corruptions).

A.1.5 Distance Properties Of Convolutional Codes.

The performance of any convolutional code depends on the decoding algorithm used and the distance properties of the code.

A.1.5.1 Minimum Free Distance d_{free}.

The minimum free distance d_{free} is the most important distance measure for convolutional codes and is the minimum distance between any two codewords. Hence, d_{free} is the weight of the minimum-weight non-zero codeword of any length. It is also the weight of the minimum-weight path in the state diagram that diverges from and remerges with the all-zero state exactly once. Because d_{free} is the minimum distance between any two codewords of any length it indicates the most likely error pattern and can be used to determine the most likely error probability.

A.1.5.2 Column Distance Function.

The Column Distance Function (CDF), d_i, is a measure of the distance properties of truncated codewords. Its name is derived from the fact that the value of d_i depends only on the first $n(i+1)$ columns of the generator matrix.

d_i is the minimum-weight non-zero codeword over the first $(i+1)$ time units and is a monotonically non-decreasing function of i. When $i=m$, the memory order of the code, d_m is called the minimum distance of the code. As $i \to \infty$ then:

$$\lim_{i \to \infty} d_i$$

represents the minimum-weight non-zero codeword of any length. It is the same as the definition for d_{free} for

noncatastrophic codes and eventually reaches d_{free} when i is approximately 3 to 4 times m the encoder memory.

The CDF is particularly important in so called sequential decoders. Sequential decoders work by *feeling* their way through a decoding tree. They measure how well they are doing by keeping an eye on the similarity between the estimate they are producing and the received sequence. If the CDF grows quickly the difference between any two codewords over any length will also grow quickly and the sequential decoder will quickly realise that a decoding error is likely.

A.1.6 Appendix Summary.

This chapter has examined the basic properties of convolutional codes. Methods of describing the codes using mathematical notation was presented and it was seen that that the notation becomes cumbersome for anything but simple codes. An alternative description using vector and matrix notation was much easier to handle. The structural properties of convolutional codes were examined and a method of obtaining a code's distance properties was described. In the final section we looked at the importance of various distance measures for codes.

B.1. Maximum Likelihood Decoding Of Convolutional Codes.

B.1.1 Introduction.

To explain the decoding of convolutional codes it is helpful to reconsider the encoding operation in terms of a *trellis diagram*. The trellis diagram is similar to the state diagram in that it shows the various states of the encoder and details of the transitions between states but it also includes timing information. Having described the trellis diagram, the decoding operation will be presented and it shall be seen that the trellis diagram can be used to describe the decoding steps required.

B.1.2 Encoding Revisited And The Trellis Diagram.

In the trellis diagram a *state* is a representation of the m newest digits in the encoder. The oldest k digits in the encoder are ignored as these will be shifted out of the encoder in the next encoding operation and therefore play no part in the generation of subsequent codewords. States are linked from left to right by branches. The branches leaving a state are normally ordered so that the uppermost branch represents an input of the 'lowest-value' sequence to the encoder (i.e. 0) and the lowest branch represents an input of the 'largest-value' sequence (i.e. M^k-1). On each branch is shown the input sequence (in brackets) that would cause the encoder to follow that branch together with the sequence of symbols

that would be output by the encoder. The branch terminates on a state that results from the given input sequence. To clarify this consider the binary encoder shown in figure B-1.

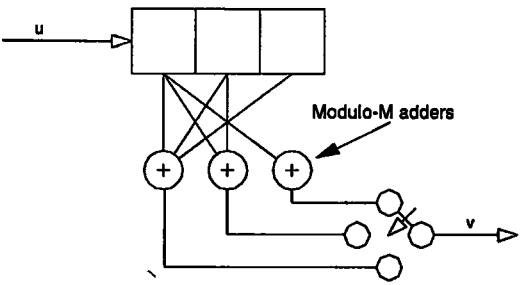


***Figure B-1* An example binary encoder.**

In this binary encoder there are 4 possible states. This is because digits are input one at a time and only the two most recent binary digits, together with the next input digit, contribute to the next encoding operation. Because the encoder only accepts one binary digit at a time, there can only ever be one of two possible inputs (0 or 1) and the corresponding trellis diagram can only have two branches leaving each state. The trellis diagram for this encoder for a single time unit is shown in figure B-2.

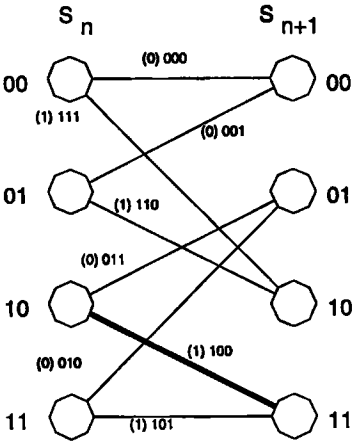


***Figure B-2* A single stage in the trellis diagram.**

In figure B-2 each state on the left is linked to states on the right by branches. Each branch is labelled with both the input, in brackets, that caused the transition and the codeword that was generated by the resulting encoder contents. As an example, the highlighted path links state '10' with state '11' because the input of a single '1' forces the encoder to hold '11' in its 2 left-most positions (previously held '10'). The fact that a '1' caused this transition is indicated by the '(1)' and the generated codeword was 100.

When many encoding operations need to be considered then many single trellises are joined, side by side, to represent all the possible states that the encoder can enter. The encoding of a block of data can then be represented by tracing a path through the trellis diagram

and the output determined by the values shown on each branch.

If a sequence of length L=5 binary digits is applied to the encoder in figure B-1, and the encoder starts from the all-zero state, then not all states may be reached for the first m=2 time units. After this initial period it is possible that the encoder could enter any of the 4 possible states. To complete the encoding operation the encoder is returned to the all-zero state by applying an input sequence of m zeros. As the encoder returns to the all-zero state the number of states it can enter is again limited. This is illustrated in figure B-3.

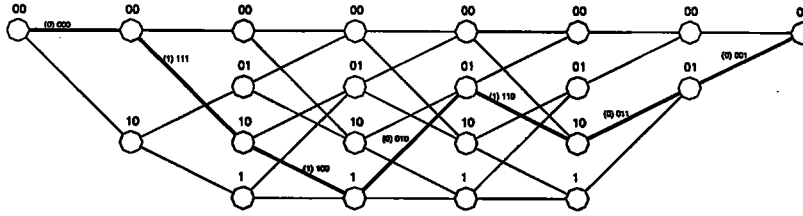


Figure B-3 **A cascaded trellis diagram.**

The highlighted path in figure B-3 represents the input of the information sequence $u=(0,1,1,0,1,0,0)$ to the encoder shown in figure B-1. The final two zeros in the sequence are used to return the encoder to the all-zero state. The codewords produced by this input sequence are seen in the codewords appearing on the highlighted branches and

are v={000,111,100,010,110,011,001}. The last two codewords are produced by the final two zeros in the input.

With regard to the definition of k and m above. For a general (n,k,m) encoder employing an alphabet of M symbols and having an input sequence of length kL symbols there will be:

1) M^m states.

2) M^k branches entering and leaving each state.

3) M^{kL} distinct paths through the trellis.

B.1.3 Decoding And The Viterbi Decoder.

The Viterbi decoder is called a Maximum Likelihood Decoder (MLD) because it chooses a path through a coding trellis that maximises the *log-likelihood function* (Viterbi 1967). The log likelihood function is a metric (a measure) of how well a chosen path through the trellis matches the received sequence.

If the information sequence $u = \{u_0 u_1 \ldots u_{L-1}\}$ were applied to an encoder the resulting codeword $v = \{v_0 v_1 \ldots v_{L+m-1}\}$ would be output. Here, the individual inputs $u_i = \{u_0 u_1 \ldots u_k\}$ and the individual codewords $v_j = \{v_0 v_1 \ldots v_n\}$.

A channel whose output consists of statistically independent discrete symbols (i.e. the symbol received

during a given interval is independent of the symbols received before it) is referred to as a Discrete Memoryless Channel DMC. 'Transmitting $\boldsymbol{v}$ over a binary input q-ary output Discrete Memoryless Channel (DMC) would result in the reception of another sequence $\boldsymbol{r}$. Representing these various sequences in terms of their individual symbols yields:

$$\boldsymbol{u} = \{u_0\, u_1 \ldots u_{kL-1}\}$$

$$\boldsymbol{v} = \{v_0\, v_1 \ldots v_{N-1}\}$$

$$\text{and } \boldsymbol{r} = \{r_0\, r_1 \ldots r_{N-1}\}$$

$$\text{where } N = n(L+m).$$

The aim in decoding the received sequence is to produce an estimate for $\boldsymbol{v}$, $\hat{\boldsymbol{v}}$, based on $\boldsymbol{r}$. A MLD chooses the $\hat{\boldsymbol{v}}$ that maximises the log-likelihood function $\log P(\boldsymbol{r}|\boldsymbol{v})$. For a DMC:

$$P(\boldsymbol{r}|\boldsymbol{v}) = \prod_{i=0}^{N-1} P(r_i|v_i)$$

and therefore:

$$\log P(\boldsymbol{r}|\boldsymbol{v}) = \sum_{i=0}^{N-1} \log P(r_i|v_i)$$

$\log P(r|v)$ is called the *metric* associated with the path $\boldsymbol{v}$ and is denoted $M(r|v)$; it follows from this that:

$$M(\boldsymbol{r}|\boldsymbol{v}) = \sum_{i=0}^{N-1} M(r_i|v_i)$$

The $M(r|v)$ are called the path metrics and the $M(r_i|v_i)$ are called the bit metrics. A partial path metric for the first j branches is denoted:

$$M\left([r|v]_j\right) = \sum_{i=0}^{j-1} M(r_i|v_i)$$

The Viterbi algorithm takes the sequence r and finds the path through the trellis that has the best metric. It does this by examining all of the branches entering a state after each time unit, and for each branch computes its partial path metric up to that state. This is done by adding the current *branch* metric to the partial path metric associated with the state that the branch emerged from. When all the partial metrics for each of the branches entering a state are known, the Viterbi algorithm rejects all but the branch with the best metric. For a given instance in time this is repeated for all the states in a vertical line in the trellis diagram. The process repeats for each time unit until all of the received sequences r_i have been examined and the extreme right-hand end of the trellis is reached. When the process is complete and the encoder has returned to the all-zero state there will be just one sequence left. This *survivor* sequence is output by the decoder as an estimate for the transmitted vector $\hat{v}$. If errors have caused some survivor sequences not to terminate on the all zero state the encoder chooses the survivor sequence with the best metric as its estimate $\hat{v}$ for v.

When decoding long sequences of data it is not practical to continue to extend the length of each survivor sequence until the last n-bit sequence has been received. This problem can be overcome by truncating the length of the survivor sequence and outputting decoded data before all the data has been received. The optimum truncation length to use has been studied by considering how long it takes for a decoder to recover from an error event (Leonard & Rodger 1989). This enables the decoder to delay its decision as to which bit it should output until that segment of the survivor sequence is no longer influenced by any previous error events. The exact length is dependent on the distance properties of the code being used and it is generally accepted that by truncating the survivor sequence to a length of around $5m$ results in little degradation of the decoding algorithm.

A chosen path $\hat{v}$ that maximises:

$$M(\mathbf{r}|\mathbf{v}) = \sum_{i=0}^{N-1} P(r_i|v_i)$$

also maximises:

$$\sum_{i=0}^{N-1} c_2\left[\log P(r_i|v_i) + c_1\right]$$

Therefore, the bit metrics log $P(r_i|v_i)$ can be replaced by $c_2[log\ P(r_i|v_i)+c_1]$ without influencing the choice of $\hat{v}$. If c_1 and c_2 are carefully chosen then all the new values can be approximated to integers with very little error and

the implementation of the algorithm becomes much easier. This approximation to integers results in sub-optimum performance of the algorithm but the degradation is slight.

B.1.4 Performance Bounds For Convolutional Codes.

In this section the generating function, introduced in chapter 3, is used to obtain an estimate of a given code's performance by assuming that the all-zero sequence is transmitted.

A decoding error occurs when a received sequence looks 'more like' one of the non-zero codewords than it does the all-zero sequence. Take as an example the code generated by the encoder shown in figure B-1. The generating function for this code is:

$$T(X) = 2X^6 + X^8 + 5X^{10} + 9X^{12} + 18X^{14} + \ldots$$

Assuming, without loss of generality, that the all-zero sequence is transmitted, a decoding error is said to occur if the all-zero path is rejected in favour of one of the other non-zero paths enumerated by the generating function. As an example, if at a given instance one of the weight 6 paths were being compared to the all-zero path then an error would definitely occur if there were 4 or more 1's in the six positions associated with the weight 6 path and with probability 0.5 if there were 3

1's in these six positions. In general, the probability of an error of this type occurring is given by (Lin & Costello 1983):

$$P_d = \begin{cases} \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e} p^e (1-p)^{d-e} & \text{d odd} \\ \frac{1}{2}\binom{d}{d/2} p^{\frac{d}{2}} (1-p)^{\frac{d}{2}} + \sum_{e=\frac{d}{2}+1}^{d} \binom{d}{e} p^e (1-p)^{d-e} & \text{d even} \end{cases}$$

An error begins when one of the paths diverging from the all-zero path begins to accumulate a better metric than the correct all-zero path. The error is complete and constitutes an *event-error* if the non-zero path remerges with the all-zero path and still has a better metric. At this point the decoder is unable to draw any more information from the received code sequence that will rectify the mistake. Assuming the transmitted sequence is of infinite length then this event-error can span anything from d_{free} time units to infinity. Therefore when calculating the probability of an event-error it is necessary to take account of the fact that any of the paths enumerated by the generating function may have been responsible for the event error. Thus:

$$P(E) < \sum_{d=d_{free}}^{\infty} A_d P_d$$

where A_d is the number of paths with gain d and P_d is the probability of the all-zero path being rejected in favour of the weight d path.

This bound can be further simplified by noting that for d odd:

$$\begin{aligned} P_d &= \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e} p^e (1-p)^{d-e} \\ &< \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e} p^e (1-p)^{\frac{d}{2}} \\ &= p^{d/2}(1-p)^{\frac{d}{2}} \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e} \\ &< p^{d/2}(1-p)^{\frac{d}{2}} \sum_{e=0}^{d} \binom{d}{e} \\ &= 2^d p^{\frac{d}{2}} (1-p)^{\frac{d}{2}} \end{aligned}$$

It can also be shown that this is an upper bound on P_d when d is even. Hence:

$$P(E) < \sum_{d=d_{free}}^{\infty} A_d \left[2\sqrt{p(1-p)}\right]^d$$

and since

$$T(X) = \sum_{d=d_{free}}^{\infty} A_d X^d$$

then

$$P(E) < T(X)\Big|_{X=2\sqrt{p(1-p)}}$$

For small p this bound is dominated by its first term (i.e. the free distance term) and the event error probability can be approximated to:

$$P(E) \cong A_{d_{free}} \left[2\sqrt{p(1-p)}\right]^{d_{free}} \cong A_{d_{free}} 2^{d_{free}} p^{\frac{d_{free}}{2}}$$

The above equations can be modified to provide an estimate for the *bit error probability*, $P_b(E)$. This is achieved by weighting each of the event error probabilities by the total number of non-zero information bits associated with each non-zero path and dividing the result by k, the number of information bits per unit time. Hence:

$$P_b(E) < \frac{1}{k} \sum_{d=d_{free}}^{\infty} B_d P_d$$

where B_d is the total number of non-zero information bits on all weight d paths.

The generating function can be modified, as described in chapter 3, so that it reflects not only the weights of the non-zero codewords but also specifies the weight of the information sequence giving rise to the codeword. The generating function is then defined as:

$$T(X,Y) = \sum_i \sum_j A_{i,j} X^i Y^j$$

where $A_{i,j}$ = number of paths with codewords of weight i and information sequences of weight j. Now,

$$\left.\frac{\partial T(X,Y)}{\partial Y}\right|_{Y=1} = \sum_{i=d_{free}}^{\infty} \sum_{j=1}^{\infty} j A_{i,j} X^i = \sum_{i=d_{free}}^{\infty} B_i X^i$$

where

$$B_i = \sum_{j=1}^{\infty} jA_{i,j}$$

substituting this into

$$P_b(E) < \frac{1}{k} \sum_{d=d_{free}}^{\infty} B_d P_d$$

and using

$$P_d < 2^d p^{\frac{d}{2}} (1-p)^{\frac{d}{2}}$$

we get

$$P_b(E) < \frac{1}{k} \frac{\partial T(X,Y)}{\partial Y}\bigg|_{X=2\sqrt{p(1-p)},Y=1}$$

For p small the expression for $P_b(E)$ will be dominated by the minimum free distance term giving:

$$P_b(E) \cong \frac{1}{k} B_{d_{free}} \left[2\sqrt{p(1-p)}\right]^{d_{free}} = \frac{1}{k} B_{d_{free}} 2^{d_{free}} p^{\frac{d_{free}}{2}}$$

The minimum free distance d_{free} of any convolutional code is strongly influenced by the encoder memory and therefore the constraint length, n_A, of the code. The more 'history' that can be built into a code the less likely it is that errors introduced by the channel will cause an incorrect branch to be 'more like' the received sequence than the correct branch. In principle the constraint length can be increased to any size required to achieve the required level of decoder performance.

B.1.5 Appendix Summary.

In this appendix we have looked at one method of decoding convolutional codes referred to as Maximum Likelihood Decoding (MLD) and discussed an algorithm due to Viterbi to do this. We discussed the metric used in the decoding algorithm and saw that this could be approximated to integer form to simplify the implementation. Next we looked at the performance of the Viterbi decoder by using the generating function described in chapter 3. Using the generating function we showed that the probability of decoding error could be approximated to

$$P(E) \cong A_{d_{free}}\left[2\sqrt{p(1-p)}\right]^{d_{free}} \cong A_{d_{free}} 2^{d_{free}} p^{\frac{d_{free}}{2}}.$$

This was then extended and an expression for the bit error probability $P_b(E)$ was obtained

$$P_b(E) \cong \frac{1}{k} B_{d_{free}}\left[2\sqrt{p(1-p)}\right]^{d_{free}} = \frac{1}{k} B_{d_{free}} 2^{d_{free}} p^{\frac{d_{free}}{2}}$$

C.1. Sequential Decoding Of Convolutional Codes.

C.1.1 Limitations Of The Maximum-Likelihood Decoder.

The main factor affecting the performance of a convolutional code is its minimum free distance, d_{free}. The minimum free distance is directly related to the encoder's memory, m, in that the larger the value of m the larger the value of d_{free} can be. In principle it is possible to increase the value of m until the required code performance is achieved provided the associated delays can be tolerated.

The Viterbi Maximum Likelihood Decoder (MLD) described in appendix B needs to keep a record of a number of survivor sequences and these must be stored in the decoder's memory. This storage requirement can sometimes become excessive and exceed practical limits. Anderson (1989) examined this problem in detail and proposed alternative decoding strategies to reduce the size of the search and storage requirements.

Consider a code with the following generalised parameters:

i) code alphabet consisting of q symbols.

ii) encoder memory = m.

As noted in appendix B section B.1.3, the length of the survivor sequences may be truncated to approximately $5m$ without significantly degrading the decoder's performance. There will be one survivor sequence for each state that the encoder can enter and if m is the memory of the decoder there will be M^m states. As a result the encoder needs to store $5mM^m$ symbols.

Another difficulty with the Viterbi MLD is the number of computations that must be performed at each stage of decoding:

- Each state will require M^k comparisons and 1 addition to extend the survivor sequence to the next state.
- Each comparison will require the formulation of a metric based on the n symbols in each input word.
- There will be M^m states to be examined.
- For a block of length $L \cdot n$ there will be L decoding steps.

As an example consider a code with the following parameters:

i) M=5.

ii) m=4.

iii) k=1.

iv) n=3.

v) L=100.

If an ideal decoder could be produced it would require $\left[\left(M^k n+1\right)M^m\right]L$ = 1,000,000 computations to decode the received block. The decoder would also have to store $5mM^m$ = 12,500 symbols as survivor sequences. Table C-1 shows how the storage requirements and computational requirements grow as m increases.

m	Computations	Storage
1	8000	25
2	40000	250
3	200000	1875
4	1000000	12500
5	5000000	78125
6	25000000	468750
7	125000000	2734375
8	625000000	15625000
9	3125000000	87890625
10	15625000000	488281250
11	78125000000	2685546875

Other code parameters:

q = 5.

n = 3.

k = 1.

L = 100.

***Table C-1* Storage and computational requirements of the Viterbi MLD.**

It is clear from the above figures that the resources required by the Viterbi MLD become excessive as m increases above about 8. It is even more significant when we take into account that these resources are required even if the received block has only a few errors. A better solution to the decoding problem would

be to use an adaptive algorithm that uses fewer resources to decode blocks with fewer errors and more resources as the error count increases. Sequential decoding provides this flexibility at the cost of sub-optimum decoding.

There are a number of sequential decoding algorithms: the Wozencraft sequential decoder (Wozencraft & Reiffen 1961), the stack or ZJ algorithm (Zigangirov 1966, Jelinek 1969), and the Fano sequential decoder (Fano 1963). In this appendix we shall be looking at the Fano sequential decoder.

C.1.2 The Fano Sequential Decoder.

The Fano sequential decoder performs the decoding task by tentatively feeling its way through the decoding tree, seeking ways forward that allow progress without undue increase of a path metric. The easiest way to see this is to consider a section of the decoding tree as shown in figure C-1 below.

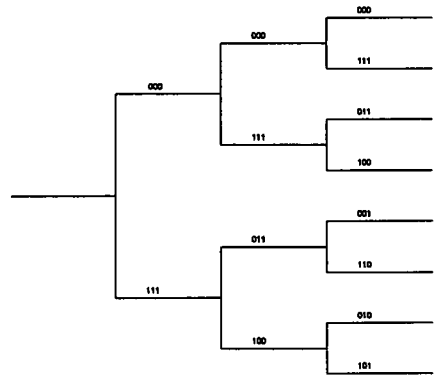


Figure C-1 **Convolutional decoding tree.**

The algorithm starts at the extreme left-hand side of the decoding tree and examines all branches that diverge from this point. It chooses, as its way forward, the branch that matches most closely the first received word. It then updates a path metric by adding the metric for the chosen branch to it (the path metric is initially set to zero). At the next node the same operation is performed and the path metric again updated. If the path metric exceeds some threshold, indicating that the overall estimate is quite different from the transmitted sequence, the decoder will move back one branch and explore other branches steming from the new node. If a better path is found that does not exceed the threshold the decoder can proceed along this alternative path. If a better path cannot be found the decoder is forced to back-track further and to explore other possible branches. If the decoder is forced too far back such that the difference between the current threshold and the path metric is larger than some set level the threshold level is allowed to increase to a higher value before back-tracking is enforced. Once the decoder starts making forward progress again the threshold is tightened to keep the estimate as tight as possible. To prevent the decoder from re-tracing the same path over and over again after back-tracking has occurred a flag is used to indicate when the decoder is examining previously traced paths. The Fano decoding algorithm is shown below together with notes on its operation.

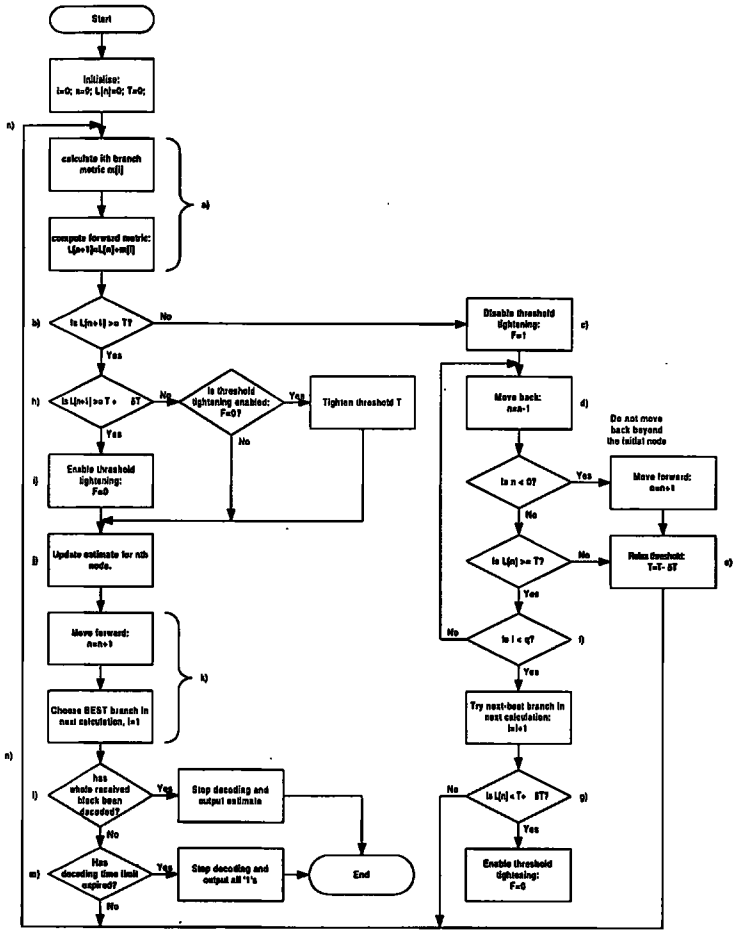


***Figure C-2* Fano sequential decoding algorithm.**

Notes on the Fano sequential decoding algorithm:

a) For each step forward the decoding algorithm has to calculate a new partial-path metric by adding the proposed branch metric to the current partial-path metric. The proposed branch will be the branch that is closest to the received sequence that has not already been rejected.

b) If the calculation causes the partial-path metric to violate the threshold, a *threshold violation* is said to have occurred. Under these circumstances the decoder starts *back-tracking*. This back-tracking takes place because the remaining branches steming from the current node will also cause a threshold violation (as they are even less like the received sequence). Hence the only way forward without relaxing the threshold is for the decoder to move back to the previous node and examine alternative paths from there.

c) When a threshold violation occurs, threshold tightening is disabled. It is disabled to stop the algorithm falling into an infinite loop where it traces a path, finds that the path causes a threshold violation (because the threshold had been tightened each time it moved forward) and back-tracks along a previously traced path.

d) With threshold tightening disabled, the decoder moves back to the node it previously visited and searches for alternative paths that do not cause a threshold violation.

e) If on moving back the partial-path metric is reduced below the current threshold, the threshold is relaxed to allow decoding to resume with this new threshold.

f) If on moving back the partial-path metric remains above the current threshold and there are branches steming from the current node that have not yet been explored then the best of these is selected as the next one to be tried.

g) Having selected the best available node for further exploration, it is necessary to re-check the partial-path metric up to this point. If the partial-path metric is above the threshold level but below the next increment point, threshold tightening must be enabled as a new path is now being traced.

h) If all went well, and a threshold violation did not occur, then the partial-path metric is examined to see if it has risen above the next increment level (the algorithm could be re-tracing an old path with a lower threshold set). If it has, and threshold tightening is enabled, then the threshold is tightened so that it is just below the current path metric.

i) If the path metric rises so that it is at least one increment above the threshold then a new path is being traced and threshold tightening is re-enabled.

j) Having chosen a branch as a likely part of the final decoded path, the information symbol associated with it is entered into the nth element of the estimate.

k) The decoder moves forward ready to examine the next set of branches and selects the best branch in preparation for this.

l) If all of the received sequence has been decoded the algorithm terminates and the estimate is output.

m) If the time taken to decode the current block of data exceeds some specified limit, then decoding is aborted and an estimate of all 1's is output(the all 1's sequence is recognised as an invalid decode).

n) If there is more to be decoded the algorithm repeats from step a).

Stage m) above indicates that decoding is aborted if the time taken to decode a block exceeds some value. This is appropriate because an excessive decoding time usually results in incorrect decoding anyway (Kallel 1988). The all 1's sequence will be recognised as an incorrect decode by the checking circuit because the Cyclic Redundancy Check (CRC) contained within the decoded block is chosen so that it is not all 1's if the data sequence is all 1's.

C.1.3 Characteristics Of Sequential Decoding.

C.1.3.1 Computational Performance.

If we let C_n equal the number of computations required to decode a block of data then C_n is a random variable and its probability distribution needs to be computed in order to determine the overall computational performance of the sequential decoder. To determine C_n consider the decoding tree shown in figure C-3.

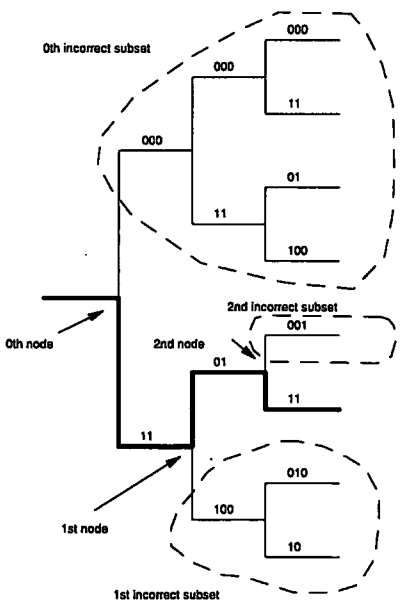


Figure C-3 **Convolutional decoding tree and incorrect subsets.**

Figure C-3 shows a number of highlighted branches representing the transmitted sequence together with the *incorrect subsets* associated with each node. To estimate the computational performance of the sequential decoder we need to consider the number of computations performed

in each of these subsets. If C_j is the number of computations performed in the jth incorrect subset then $\Pr[C_j \geq \eta] \cong A\eta^{-\alpha}$ (Savage 1966). $\Pr[C_j \geq \eta]$ is a *Pareto distribution*, α is called the *Pareto exponent* and the constant A depends on the decoding algorithm used.

Now, α is related to the code rate R by the parametric equation $R=\frac{E_o(\alpha)}{\alpha}$, where $E_0(\alpha)$ is the *Gallager function*, determined completely by the channel transition probabilities. Consequently $\Pr[C_j \geq \eta]$ is independent of the encoder memory m.

To determine the average computational performance we must calculate the moments of C_j $E[C_j^i]$:

$$\Pr[C_j < \eta] = 1 - \Pr[C_j \geq \eta] = 1 - A\eta^{-\alpha}$$

Differentiating $\Pr[C_j < \eta]$ with respect to η gives the probability density function:

$$f_{C_j}(\eta) = \frac{\partial(1 - A\eta^{-\alpha})}{\partial \eta} = \alpha A \eta^{-\alpha-1}$$

The moments of C_j can be calculated using:

$$E[C_j^i] = \int_1^\infty \eta^i f_{C_j}(\eta)\, d\eta = \alpha A \int_1^\infty \eta^{i-\alpha-1}\, d\eta$$

$$= \alpha A \frac{\eta^{i-\alpha}}{i-\alpha}\Bigg|_1^\infty = \frac{\alpha A}{i-\alpha} \lim_{\eta \to \infty} \{\eta^{i-\alpha} - 1\}$$

For the ith moment to be finite α must be greater than i. The mean is obtained when i = 1 and therefore if $\alpha \leq 1$ the average number of computations per branch is unbounded and sequential decoding becomes impractical. The significance of α = 1 is that it specifies the limit of practical decoding and consequently $R=\frac{E_o(1)}{1}=E_0(1)$ is referred to as the *computational cut-off rate* R_0. Kallel(1988) has examined sequential decoding when combined with code combining and found that the effect of code combining is to increase R_0. This increase is a result of a change to the 'effective channel' which has the effect of changing the apparent channel transition probabilities.

For $R \rightarrow R_0$ the performance of sequential decoding is approximately equal to that of the Maximum Likelihood Decoding and a slight increase in the value of m can compensate for the degradation without any effect on $E[C_j]$.

For $R < R_0$ the performance of sequential decoding is sub-optimum but again the performance can be improved by sufficiently increasing the value of m provided the increased complexity can be tolerated.

C.1.3.2 Other Factors Effecting Performance.

The choice of metric can also effect a decoder's performance. Consider the Fano metric defined by (Lin & Costello 1983):

$$M(r_i|v_i)=\log_2\frac{P(r_i|v_i)}{P(v_i)}-nLR$$

The value of nLR achieves a balance among bit error probability $P_b(E)$, erasure probability $P_{erasure}$ and $E[C_j]$. In particular the Fano metric is normally chosen to give appropriate weights to paths of differing lengths. As an example, a path of length 20 with 4 errors should have roughly the same metric as a path of length 10 with only 2 errors. Reducing the value of nLR would bias the metric in favour of the longer path giving rise to increased $P_b(E)$ but resulting in less searching. If on the other hand the value of nLR were increased, the bias would be in favour of the shorter paths. The consequence of this would be a better $P_b(E)$ but a higher value for $P_{erasure}$ due to the increase in searching.

The Column Distance Function (CDF) is a measure of the distance between the closest two paths over any truncated segment of the code tree. If

$$x=(x_{00},x_{01},..,x_{0n},x_{10},x_{11},..,x_{1n},...,x_{L0},x_{L1},..,x_{Ln})$$

and

$$v=(v_{00},v_{01},..,v_{0n},v_{10},v_{11},..,v_{1n},...,v_{L0},v_{L1},..,v_{Ln})$$

represent the two closest sequences corresponding to two different paths of length $L+1$ branches through the decoding tree, then the CDF at distance $L+1$ is the distance between x and y.

The CDF plays a major part in determining the performance of a sequential decoding algorithm as it dictates how quickly the metric used in decoding will deteriorate if an incorrect path is being traced (Chevillat & Costello 1976). The faster the growth in the CDF the quicker the decoder will cause a threshold violation if an incorrect path is being traced and, consequently, the amount of searching will be reduced. This reduction in decoding time will reduce the probability of buffer overflow reducing the number of erasures. In the case where the decoder uses a time limit before giving up on decoding and requests the transmitter to re-send the block, the effect will be to reduce the number of re-transmissions.

C.1.4 Appendix Summary.

In this appendix we began by looking at the practical limitations of maximum likelihood decoding and concluded that it became impractical for encoder memories, m, greater than about 8. We then looked at a practical implementation of the Fano sequential decoder and discussed its operation with the aid of a flow chart. The computational requirements of the Fano decoder were

discussed next and the importance of the *computational cut-off rate* was explained. We concluded this section by looking at other factors that effect the decoder's performance and noted that the CDF is an important parameter for sequential decoders as it influences the amount of searching that the decoder performs.

D.1. A Detailed Description Of The Scheme Developed.

Figures D-1 and D-2 describe the Encode/Transmit and Receive/Decode schemes respectively in terms of Data Flow Diagrams. A key to the symbols used is shown in figure D-3 and each element in the scheme is described in detail in the following sections.

The description of each element takes the following format:

1) A detailed description of the necessary processing and the circumstances under which this is necessary.
2) Diagrams depicting the storage and transfer of data.
3) A logical list of the steps necessary (without reasons) for the task to be successfully completed.
4) A structured English description of the code required for each process.

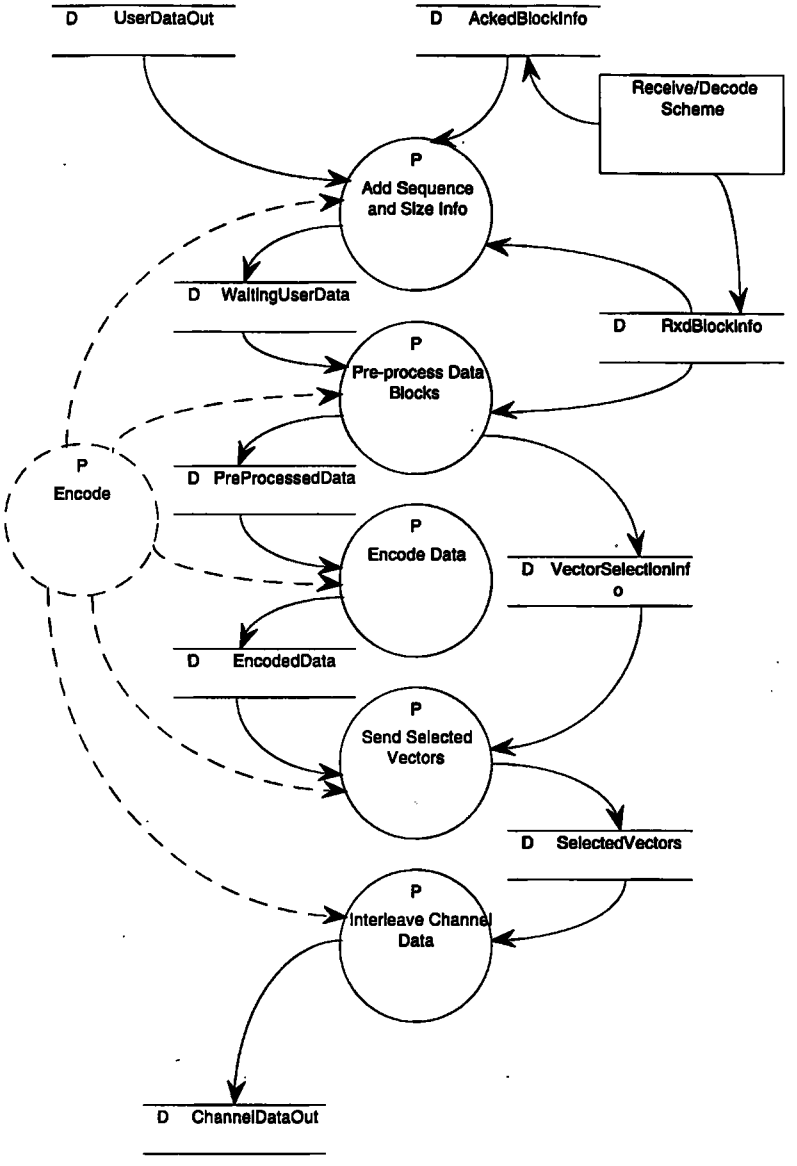


Figure D-1 **Encode/transmit scheme data flow diagram.**

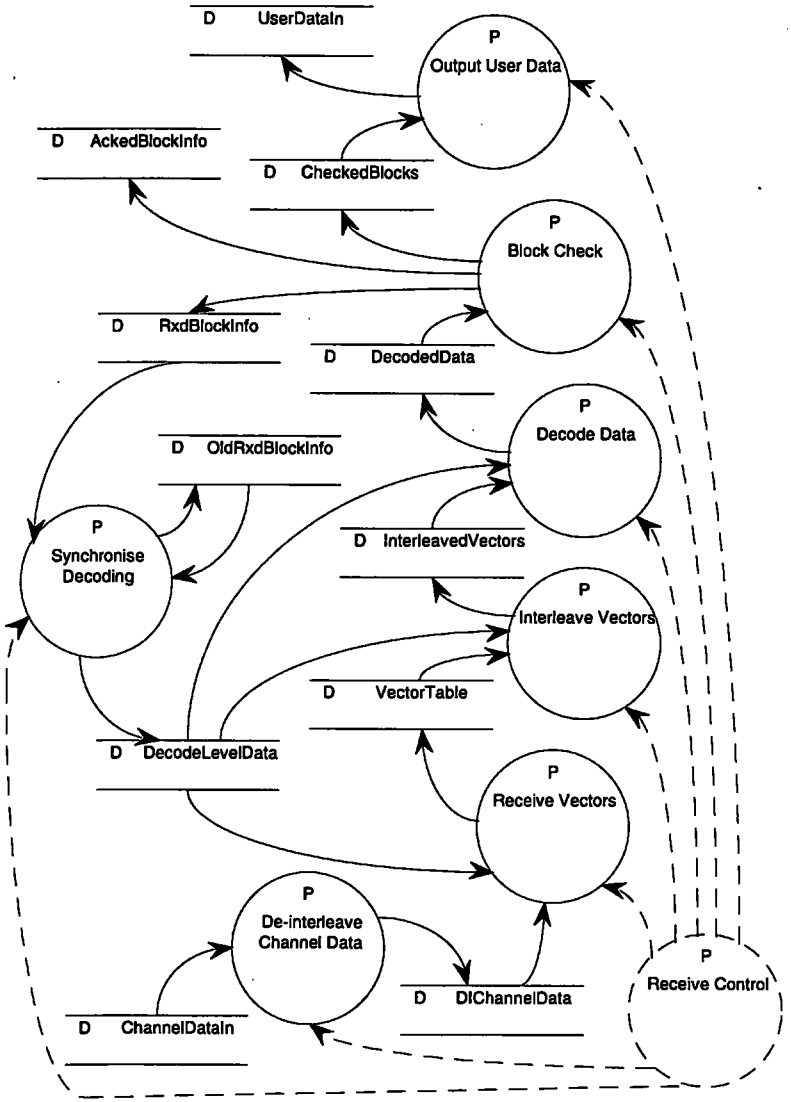
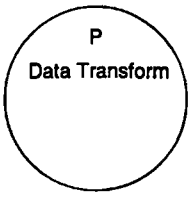
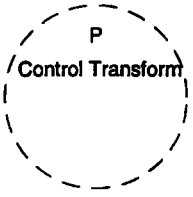


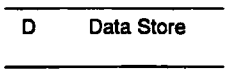
Figure D-2 **Receive/decode scheme data flow diagram.**



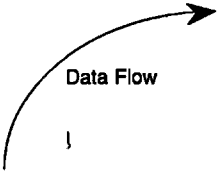
Data Transform - Receives data and event/trigger commands. Sends processed data and event/trigger commands.



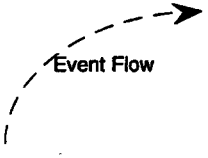
Control Transform - Controls the processes it is linked to according to a state transition table and event stimulus.



Data Store - Where data "rests" when it is neither flowing nor being operated on.



Discrete Data Flow - Shows the flow of data around the system (data may also flow in the opposite direction but the arrow indicates the nett flow)



Event Flow - Shows the origin and source of commands. These are typicaly generated by a control process to 'kick' a data process into action (there may be event/trigger information flowing in the opposite - e.g. complete - but only the nett flow is shown).

Figure D-3 **Key to symbols used in data flow diagrams.**

D.1.1 Encode/Transmit Scheme.

D.1.1.1 Add Sequence And Size Info.

If there is a free slot in the *WaitingUserData* circular buffer this process takes the next block of upto 100 symbols from the *UserDataOut* circular buffer. It appends length information (in duplicate) to indicate how much of the block contains valid user data and if there are less than 100 symbols in the *UserDataOut* store the data portion is padded with zeros.

The process also appends a number (in duplicate) that cycles from 0 to 215 to act as sequencing information which is derived from *AckedBlockInfo*. *AckedBlockInfo* is maintained by the local receive/decode scheme and if it contains the value **N** then this indicates that the local receive/decode scheme has successfully decoded blocks with the value **N** in their ACK-fields. This indicates that the remote system has received and successfully decoded all blocks upto and including block **N**. The blocks in the *WaitingUserData* frame have consecutive sequence numbers that have been derived from previous values in *AckedBlockInfo*. If the value in *AckedBlockInfo* is greater than or equal to any of the values in the SEQ-fields of blocks in *WaitingUserData* then those blocks are removed from *WaitingUserData* to make room for more new blocks. If the value in *AckedBlockInfo* is lower than any

of the values in the SEQ-fields of blocks in *WaitingUserData* then it is necessary to re-sequence the SEQ-fields because a sequencing error has occurred (see chapter 6 section 6.4.1.3 and section D.1.2.5 below). To re-sequence the SEQ-fields, set the SEQ value in the block occupying slot 0 of *WaitingUserData* to one greater than the value in *AckedBlockInfo* and renumber the subsequent blocks with incremental values.

An AGE is associated with each block so that the scheme can keep track of how long a particular block has been in the system. The resulting block of information is stored in the next free slot of the *WaitingUserData* circular buffer.

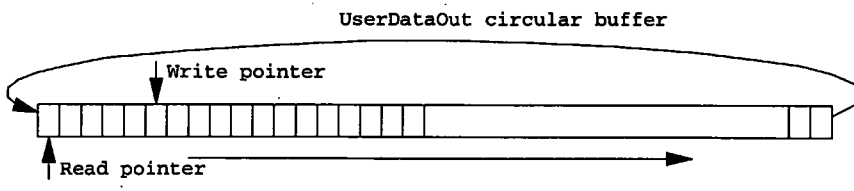


***Figure D-4a* Structure of *UserDataOut* circular buffer.**

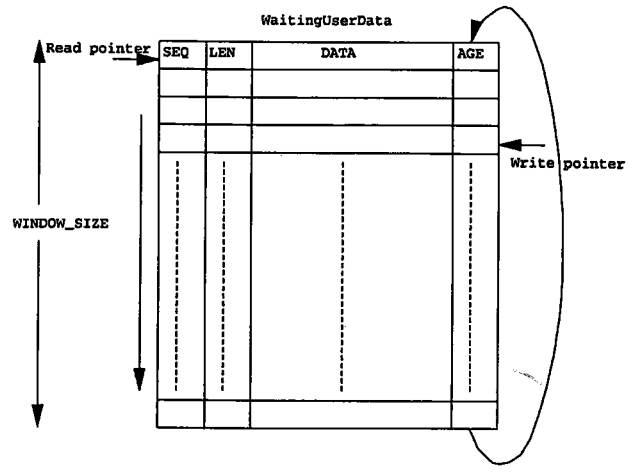


***Figure D-4b* Structure of *WaitingUserData* circular buffer.**

Steps in 'Add Sequence and Size Info':

- Examine *AckedBlockInfo* to determine which blocks have been acknowledged.
- Read (invalidate) the acknowledged blocks from *WaitingUserData* to free up space.
- While there is space in *WaitingUserData*:
    1. Get next block of data (DATA) from *UserDataOut* (there may be none - length 0).
    2. Add Sequence (SEQ) and length (LEN) information.
    3. Set AGE=0 for each new block (this indicates a brand new block).
    4. Transfer block to *WaitingUserData*.
- If necessary re-sequence SEQ-fields in *WaitingUserData*

Structure of 'Add Sequence and Size Info' Function.

AdSeqSiz():

Read value in *AckedBlockInfo*.
Read value in the ACK-field of the block in slot 0 of *WaitingUserData* (= WUD[0].ACK).
If *AckedBlockInfo* >= WUD[0].ACK and *AckedBlockInfo* <= WUD[window_size-1].ACK
BEGIN(1A)
    Remove blocks whose ACK-fields have a value lower than *AckedBlockInfo* by reading them from *WaitingUserData*.
    While there are free slots in *WaitingUserData*
    BEGIN(2)
        Get upto 100bytes from *UserDataOut* and transfer them to the DATA-field of temporary storage.
        Write number of bytes read into LEN-field of temporary storage
        Write 0 into AGE-field of temporary storage
        Get *WaitingUserData* write pointer (= WUD.wp)
        Determine ACK-field value in block at WUD[WUD.wp - 1].ACK
        Increment value and store in ACK-field of temporary storage.
        Write contents of temporary storage to *WaitingUserData*
    END(2)
END(1A)
If *AckedBlockInfo* < WUD[0].ACK - 1
BEGIN(1B)
    Resequence values held in ACK-fields of blocks in *WaitingUserData* starting with *AckedBlockInfo* + 1.
END(1B)

D.1.1.2 Pre-process Data Blocks.

The data in *WaitingUserData* represents those blocks that have not yet been successfully transmitted. This routine takes the value in *RxdBlockInfo* and copies it into the ACK-field of each block in *WaitingUserData*. Doing this ensures that the far-end is told which of its blocks the local receive scheme (see section D.1.2 below for a description of this) has successfully received. The routine also calculates a 7-symbol CRC word based on each block in *WaitingUserData* which is copied to the corresponding CRC-field. Each of the resulting blocks is copied to *PreProcessedData*.

After encoding, the actual data that gets transmitted depends on how may attempts have been made to send a particular block. The number of attempts is directly related to the length of time a block remains in *WaitingUserData* which is indicated by the AGE-field. This routine updates the AGE-field in *WaitingUserData* and copies the new values to an array called *VectorSelectionInfo*. *VectorSelectionInfo* is used by a later routine to ensure that transmission of the correct blocks takes place.

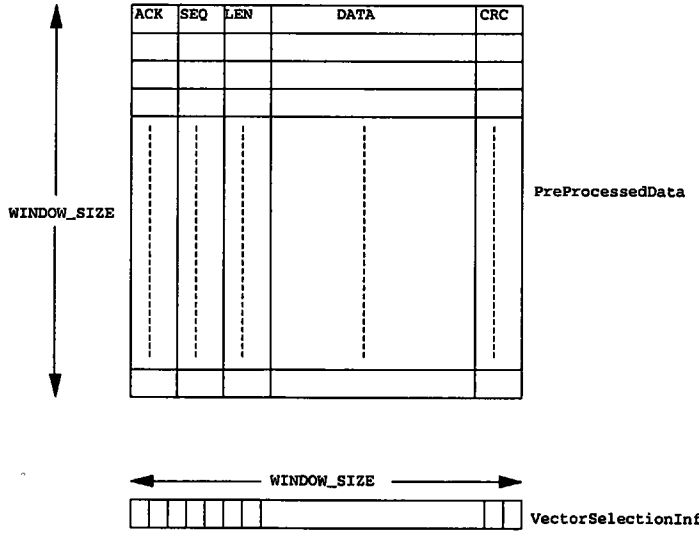


***Figure D-5* Structure of PreProcessedData and VectorSelectionInfo stores.**

Steps in Pre-process Data Blocks:

- Read *RxdBlockInfo* and copy the value into the ACK-field of each block.

- Calculate the CRC for each block and copy the value into the CRC-field of each block.
- Copy each block with ACK, SEQ, LEN, DATA and CRC fields to *PreProcessedData*.
- Update the AGE-field for each block in *WaitingUserData* (0→1→2→3→1→1→...→1).
- Copy each AGE-field, in order, to the successive locations in *VectorSelectionInfo*.

Structure of 'Pre-process Data Blocks' Function.

PreProcDataBlks():

For all blocks in *WaitingUserData*
BEGIN(1)
Copy SEQ, LEN and DATA-fields into temporary storage.
Copy value in *RxdBlockInfo* to the ACK-field in temporary storage.
Calculate CRC value based on data in ACK, SEQ, LEN and DATA fields and store in CRC-field of temporary storage.
Copy contents of temporary storage to the next slot in *PreProcessedData*.
END(1)
Update the values in the AGE-field of each block in *WaitingUserData*.
Rules:
Initial Settings: AGE.value=0, AGE.flag=0.
To update: If AGE.flag = 0 and AGE.value < 3 then AGE.value=AGE.value+1
If AGE.value = 3 then AGE.value = 1 also AGE.flag=1.
Copy the values in the AGE-field of each block in *WaitingUserData* **in order** to *VectorSelectionInfo*.

D.1.1.3 Encode Data.

Each block in *PreProcessedData* is encoded using a 1/3rd rate convolutional encoder. The encoding process produces 3 separate vectors, $v_i^{(1)}$ $v_i^{(2)}$ $v_i^{(3)}$, for each complete block in *PreProcessedData*. These vectors are stored in *EncodedData*.

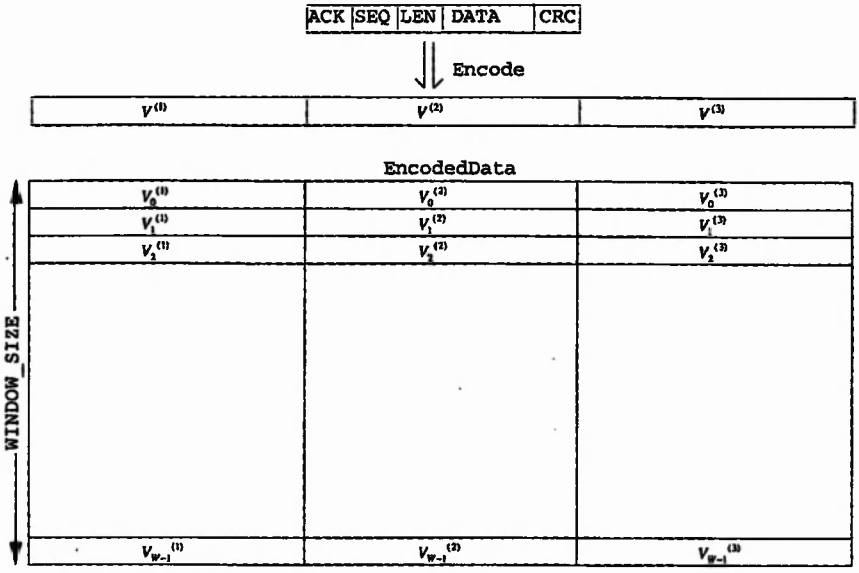


Figure D-6 **Encoding process and structure of *EncodedData* store.**

Steps in Encode Data:

Encode all blocks in *PreProcessedData* and transfer resulting vectors $v_1^{(1)}$ $v_1^{(2)}$ $v_1^{(3)}$ to *EncodedData*

Structure of 'Encode Data' Function.

EncodeData():

For each block in *PreProcessedData*.
BEGIN(1)
Encode block of data to produce $v^{(1)}$, $v^{(2)}$ and $v^{(3)}$.
Store $v^{(1)}$, $v^{(2)}$ and $v^{(3)}$ in next slot of *EncodedData*.
END(1)

D.1.1.4 Send Selected Vectors.

Send Selected Vectors copies those vectors indicated by the *VectorSelectionInfo* array to the *SelectedVector* store.

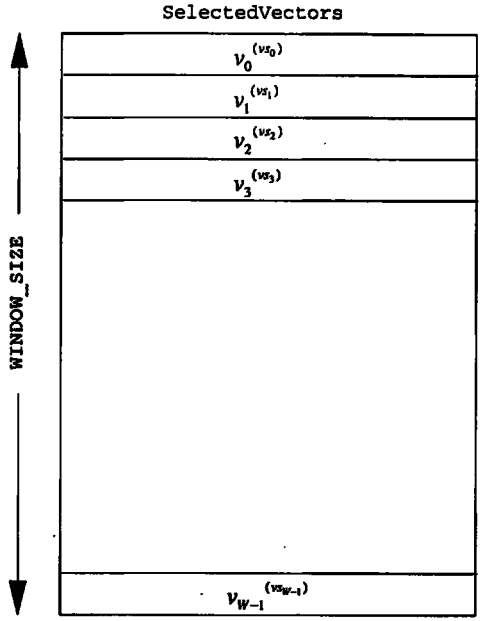


***Figure D-7* Structure of *SelectedVectors* store.**

Steps in Send Selected Vectors:

- Using the sequence of values in *VectorSelectionInfo* = $\{vs_0, vs_1, vs_2, \ldots\ldots, vs_{W-1}\}$ copy the appropriate vectors $v_0^{(vs_0)}, v_1^{(vs_1)}, v_2^{(vs_2)}, \ldots\ldots v_{W-1}^{(vs_{W-1})}$ from *EncodedData* to *SelectedVectors*.

Structure of 'Send Selected Vectors' Function.

SndSlctVect():

For each row **r** in *EncodedData* (= ED.b[r]).
BEGIN(1)
    Read next row **r** in *VectorSelectionInfo* (= VSI.d[r])
    If VSI.d[r] = 1 copy ED.b[r].v1[-] to next slot in *SelectedVectors* (= SV.d[r].b[-]).
    If VSI.d[r] = 2 copy ED.b[r].v2[-] to next slot in *SelectedVectors* (= SV.d[r].b[-]).
    If VSI.d[r] = 3 copy ED.b[r].v3[-] to next slot in *SelectedVectors* (= SV.d[r].b[-]).
END(1)

D.1.1.5 Interleave Channel Data.

When data is transmitted across the HF channel disturbances may cause a number of consecutive symbols to be corrupted. It is better for the error-detection/correction scheme if these error bursts are broken up so that errors occur in isolated positions. This is achieved by a process known as interleaving. In interleaving, a block of data to be transmitted is written to an array in rows and then read-out in columns. When the data is received at the other end it is written to an array in columns and then output to the detector/decoder in rows. The combination of these steps at each end has the effect of breaking-up error bursts. The Interleave Channel Data process described here writes the columns of *SelectedVectors* to the *ChannelDataOut* Vector.

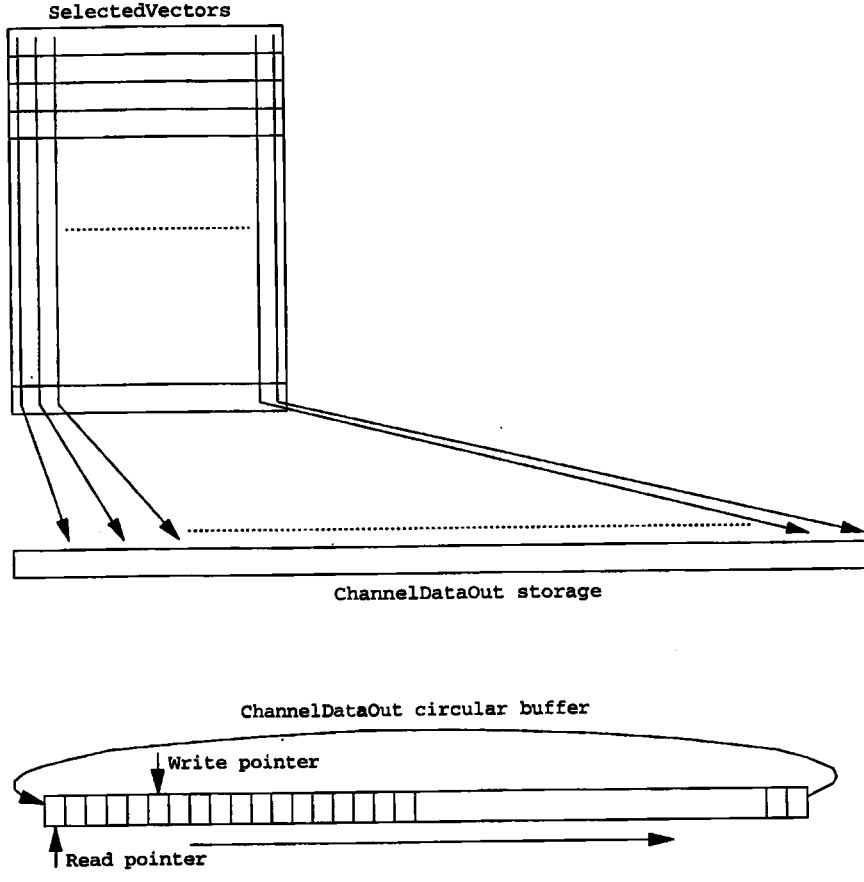


Figure D-8 **Channel Data Interleave process and structure of *ChannelDataOut* circular buffer.**

Steps in Interleave Channel Data:

- Interleave the data in *SelectedVectors* by reading it out in columns to *ChannelDataOut*.

Structure of 'Interleave Channel Data' Function.

ILChData():

For each column c in *SelectedVectors* (= SV.d[-].b[c])
BEGIN(1)
    Write SV.d[-].b[c] to *ChannelDataOut*.
END (1)

D.1.2 Receive/Decode Scheme.

D.1.2.1 De-interleave Channel Data.

The purpose of this routine is to reverse the data transformation caused by the 'Interleave Channel Data' process described above. The data is read-in from the channel and stored in the columns of *DIChannelData*

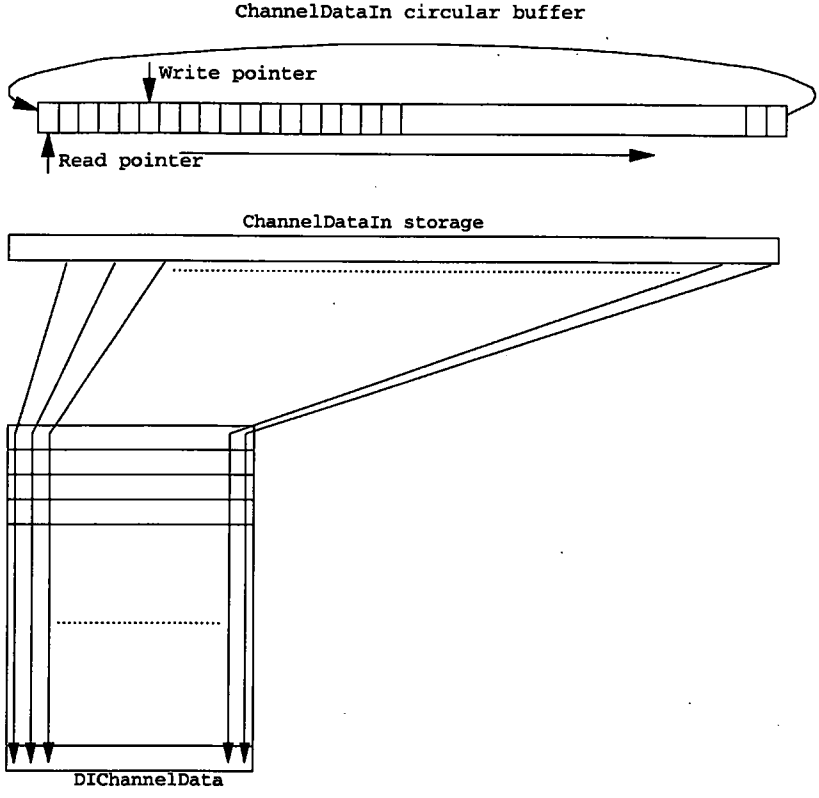


Figure D-9 **Channel Data De-interleave process and structure of DIChannelData & ChannelDataIn circular buffer.**

Steps in De-interleave Channel Data:

Read from *ChannelDataIn* into the columns of DIChannelData.

Structure of 'De-interleave Channel Data' Function.

DIChData():

For each column c in *DIChannelData* (= DICD.d[-].b[c])
BEGIN(1)
    Read next block of 'ASLDC_SIZE' bytes from *ChannelDataIn* into temporary storage.
    Write contents of temporary storage to DICD.d[-].b[c].
END(1)

D.1.2.2 Receive Vectors.

Receive Vectors takes the data in the rows of *DIChannelData* (Which in the absence of transmission errors is identical to *SelectedVectors* described above) and copies it to the appropriate location in *VectorTable*. It determines which location a particular data-block should be placed in by examining the contents of *DecodeLevelData*. As an example, if the *j*th entry in *DecodeLevelData* contains the value *i* then the *j*th block of data in *DIChannelData* is copied to the *i*th column and *j*th row of *VectorTable*.

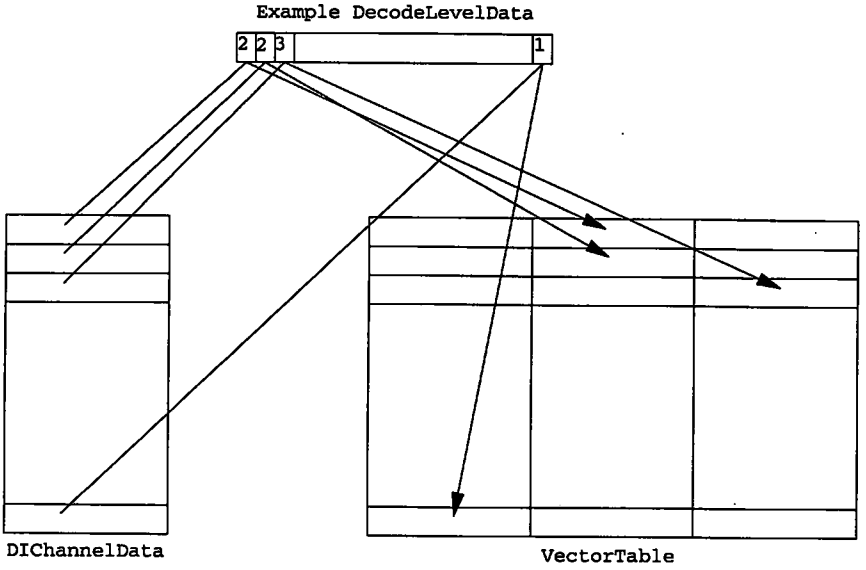


Figure D-10 **Receive Vectors Process and structure of *VectorTable*.**

Steps in Receive Vectors:

- Using the sequence of values in *DecodeLevelData* copy the row vectors in *DIChannelData* to the appropriate columns in *VectorTable*.

Structure of 'Receive Vectors' Function.

RxVectrs():

For each row **r** in *DIChannelDataIn* (= DICD.d[**r**].b[-])
BEGIN(1)
Copy DICD.d[**r**].b[-] to temporary storage.
Read rth element in *DecodeLevelData* (**c** = DLD.d[**r**]).
If **c**=1 then copy temporary storage to rth row and **1st** column of *VectorTable* (= VT.b[**r**].v1[-])
If **c**=2 then copy temporary storage to rth row and **2nd** column of *VectorTable* (= VT.b[**r**].v2[-])
If **c**=3 then copy temporary storage to rth row and **3rd** column of *VectorTable* (= VT.b[**r**].v3[-])
END(1)

D.1.2.3 Interleave Vectors.

It is important that the *Interleave* referred to here is not confused with the interleave/de-interleave process described above to break-up error-bursts that occur during transmission. Normally, when a convolutional encoder produces a 1/3rd rate code, the data at the 3 output nodes of the encoder is interleaved by a commutator before being passed-on as shown in appendix A figure A-1. In the scheme described here the blocks of data appearing on each output node of the encoder are handled separately and transferred individually. However, before the decoder can perform its task, the received data must be interleaved so that it appears as if it has just come from the output of a standard convolutional encoder. This process uses the values in *DecodeLevelData* to determine how many of the received blocks should be interleaved and stored in *InterleavedVectors* before decoding is attempted. As a specific example if the 5th entry in *DecodeLevelData* contains the value 2 then this process takes the vectors $v_5^{(1)}$ *and* $v_5^{(2)}$, interleaves them and stores the resulting vector in *InterleavedVectors*.

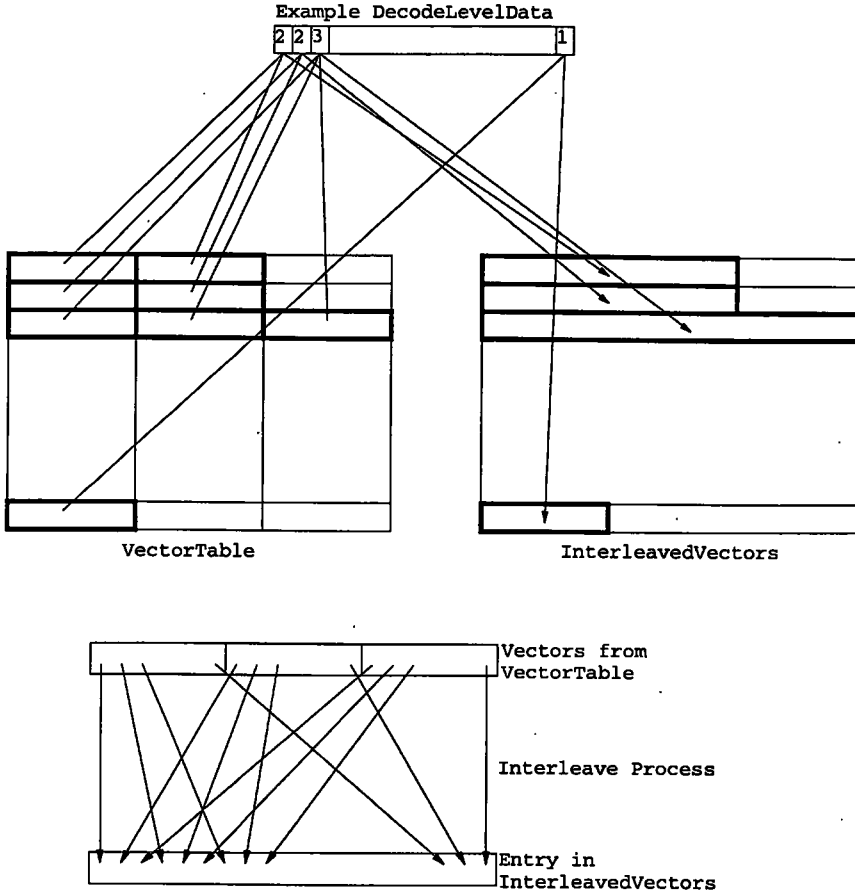


***Figure D-11* Interleave Vectors process and structure of *InterleavedVectors*.**

***Steps in Interleave Vectors*:**

- Using the sequence of values in *DecodeLevelData* take the appropriate number of vectors from *VectorTable* for each row and interleave them. Store the resulting data in *InterleavedVectors*.

Structure of 'Interleave Vectors' Function.

IlVectrs():

```
For each row r in VectorTable ( = VT.b[r].vx[-] : x = 1, 2 or 3 )
BEGIN(1)
        Read rth element in DecodeLevelData ( c = DLD.d[r] ).
        If c=1
        BEGIN(2A)
                Copy rth row and 1st column of VectorTable ( = VT.b[r].v1[-] ) to store1.
                Copy store1 to rth row of InterleavedVectors ( = ILV.b[r].d[-] )
        END(2A)
        If c=2
        BEGIN(2B)
                Copy rth row and 1st column of VectorTable ( = VT.b[r].v1[-] ) to store1.
                Copy rth row and 2nd column of VectorTable ( = VT.b[r].v2[-] ) to store2.
                Interleave contents of store1 and store2 and write result to temporary storage.
                Copy temporary storage to rth row of InterleavedVectors ( = ILV.b[r].d[-] )
        END(2B)
        If c=3
        BEGIN(2C)
                Copy rth row and 1st column of VectorTable ( = VT.b[r].v1[-] ) to store1.
                Copy rth row and 2nd column of VectorTable ( = VT.b[r].v2[-] ) to store2.
                Copy rth row and 3nd column of VectorTable ( = VT.b[r].v3[-] ) to store3.
                Interleave contents of store1, store2 & store3 and write result to temporary storage.
                Copy temporary storage to rth row of InterleavedVectors ( = ILV.b[r].d[-] )
        END(2C)
END(1)
```

D.1.2.4 Decode Data.

Decode uses the values in *DecodeLevelData* to determine which decode-level to use (1:1, 1:2 or 1:3) for each block and the size of the block to read from *InterleavedVectors*. Having obtained this information the routine produces one decoded output vector for each block read from *InterleavedVectors* and stores it in *DecodedData*.

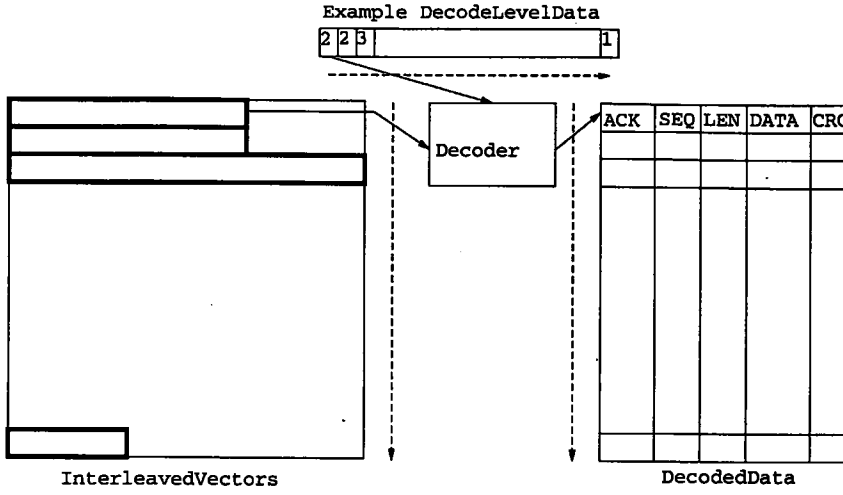


Figure D-12 **Decode process using *DecodeLevelData* to set decode parameters.**

Steps in Decode:

- Using the sequence of values in *DecodeLevelData* set the decode-level, as appropriate for each block, and decode each of the blocks in *InterleavedVectors*. Store the decoder output, for each block decoded, in the rows of *DecodedData*.

Structure of 'Decode Data' Function.

DecData():

For each row r in *InterleavedVectors* (= ILV.b[r].d[-])
BEGIN(1)
Read rth element in *DecodeLevelData* (l = DLD.d[r]).
Set decode parameters for level l decoding
Read l*(ASLDC_SIZE) bytes from row r in *InterleavedVectors* into temporary storage.
Execute decoding algorithm using temporary storage and decode parameters as input.
Copy output of decoder to rth row in *DecodedData* (= DD.b[r]):

- copy 1st 6-symbols to ACK-field (= DD.b[r].ack)
- copy next 6-symbols to SEQ-field (= DD.b[r].seq)
- copy next 6-symbols to LEN-field (= DD.b[r].len)
- copy next 100-symbols to DATA-field (= DD.b[r].dat)
- copy last 7-symbols to CRC-field (= DD.b[r].crc)

END(1)

D.1.2.5 Block Check.

'Block Check' checks the blocks in *DecodedData* to see if they have been received correctly. Correctly received blocks are stored in *CheckedBlocks* and their SEQ and ACK fields are used to update *RxdBlockInfo* and *AckedBlockInfo* respectively. An example of the use of the SEQ and ACK fields follows:

The SEQ field is information from the remote transmit site about the position of a particular block in the overall message. As an example a frame of blocks with sequence numbers 163 to 170 inclusive may have been received. If the local decode scheme manages to decode blocks 163, 164, 165, 167, 168 and 170 then these blocks will be stored in the appropriate slots in *CheckedBlocks* with their valid flags set and the slots corresponding to blocks 166 and 169 will be left blank (and their valid flags will remain clear). Also, because all blocks upto and including block 165 have now been received, the value 165 is written to RxdBlockInfo.

The remote transmit site will have written a value into the ACK field of all blocks in a frame (this corresponds to the blocks its receive scheme has managed to decode). All blocks within a frame will have the same ACK value and if this value has increased (indicating that the

remote scheme has received, decoded and stored some more blocks) then the new value is stored in *AckedBlockInfo*. A new value in *AckedBlockInfo* tells the local encode/transmit scheme that some blocks in *WaitingUserData* can be removed as these have now been received.

As discussed in chapter 6 section 6.4.1.3, certain corruptions in the ACK field need to be detected and corrected to prevent the system from locking up. The detection and correction procedure involves intelligently looking at the values in the SEQ field and if necessary adjusting the values placed in the ACK fields of outgoing blocks.

The checking procedure has the following basic rules:

- A CRC based on the first 118 symbols in the 125-symbol block should be identical to the 7-CRC symbols held in the last 7 symbols of the block.
- Both values in the ACK, SEQ and LEN fields should be identical and within specified ranges.

If these basic checks are all okay the checked data in *DecodedData* is transferred to *CheckedBlocks*, the ACK-field is used to update *AckedBlockInfo* and the SEQ-field is used to update *RxdBlockInfo*.

If any of the following checks fail the block is discarded:

- The CRC's fail to agree.
- The duplicated values in the ACK, SEQ or LEN fields fail to agree.
- The values in the SEQ or LEN fields are outside the specified ranges.
- The values in the ACK field are higher than the specified range.

If the values in the ACK field are lower than the specified range then it is possible that adjustment of the outgoing SEQ-field is required.

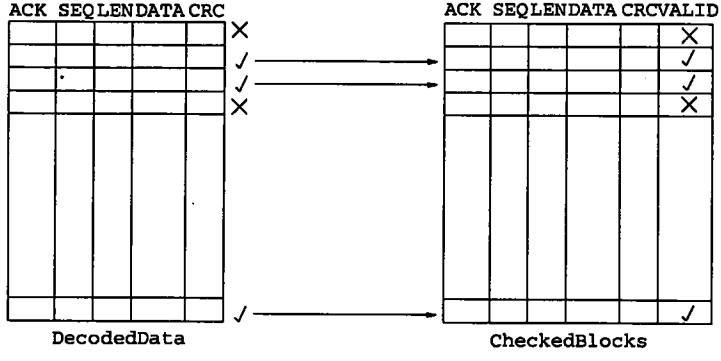


***Figure D-13* Block Check process and structure of *CheckedBlocks*.**

Steps in Block Check:

- Check each row of *DecodedData* as follows:

Re-compute CRC based on first 118 symbols of each row

If re-computed CRC equals value in CRC-field:

If both values in LEN-field are equal and within valid range:

If both values in SEQ-field are equal and within valid range:

If both values in ACK-field are equal and within valid range:

Copy checked row into the slot indicated by SEQ-field of *CheckedBlocks* and set valid flag.

Write ACK value in decoded block to *AckedBlockInfo*.

If both values in ACK-field are equal but range is **lower** than expected:

If *AckRangeFlag* is clear:

Set *AckRangeFlag*

Set *AckRangeCounter* = 1

Store value held in ACK-field in *AckErrorValue*.

If *AckRangeFlag* is set

If value in *AckErrorValue* is equal to that in the ACK-field.

Increment *AckRangeCounter*.

If *AckRangeCounter* > *Threshold*

Set *AckedBlockInfo* = *AckErrorValue*.

Copy checked row into the slot indicated SEQ-field of *CheckedBlocks* and set valid flag.

Clear *AckRangeFlag*.

If value in *AckErrorValue* is different to that in the acknowledge-field.

Clear *AckRangeFlag*.

- Write SEQ value in the last sequential block staring from slot 0 in *CheckedBlocks* to *RxdBlockInfo*.

Structure of 'Check Block' Function.

ChkBlk():

For each block in *DecodedData*
BEGIN(1)
Re-compute CRC based on first 118 symbols of block
If re-computed CRC equals value in CRC-field
BEGIN(2)
If both values in LEN-field are equal and within valid range
BEGIN(3)
If both values in SEQ-field are equal and within valid range
BEGIN(4)
If both values in ACK-field are equal and within valid range
BEGIN(5A)
Copy checked row into the slot indicated by SEQ-field of *CheckedBlocks* and set valid flag.

Write ACK value in decoded block to *AckedBlockInfo*.
END(5A)
If both values in ACK-field are equal but range is lower than expected
BEGIN(5B)
If *AckRangeFlag* is clear
BEGIN(6A)
Set *AckRangeFlag*
Set *AckRangeCounter* = 1
Store value held in ACK-field in *AckErrorValue*.
END(6A)
If *AckRangeFlag* is set
BEGIN(6B)
If value in *AckErrorValue* is equal to that in the ACK-field.

BEGIN(7A)
Increment *AckRangeCounter*.
If *AckRangeCounter* > *Threshold*
BEGIN(8)
Set *RxdBlockInfo* = *AckErrorValue*.
Copy checked row into the slot indicated by SEQ-field of *CheckedBlocks* and set valid flag.
Clear *AckRangeFlag*.
END(8)
END(7A)
If value in *AckErrorValue* is different to that in the acknowledge-field.
BEGIN(7B)
Clear *AckRangeFlag*.
END(7B)
END(6B)
END(5B)
END(4)
END(3)
END(2)
END(1)
Write SEQ value in last sequential block starting from slot 0 in *CheckedBlocks* to *RxdBlockInfo*.

D.1.2.6 Output User Data.

This routine is responsible for delivering the blocks of data successfully transferred from a remote machine to the user. It takes blocks of data from *CheckedBlocks* and transfers them to *UserDataIn*. To ensure that the user does not read blocks out of order, it will only transfer a block if all other blocks upto the one currently being examined have already gone. As an example, if all blocks upto and including block 143 have already gone and *CheckedBlocks* contains blocks 144, 145, 146, 150 and 151 then only the DATA field of blocks 144, 145 and 146 will be transferred to *UserDataIn*. Blocks 147, 148 and 149 must be successfully received and transferred to *UserDataIn* before blocks 150 and 151 can be transferred.

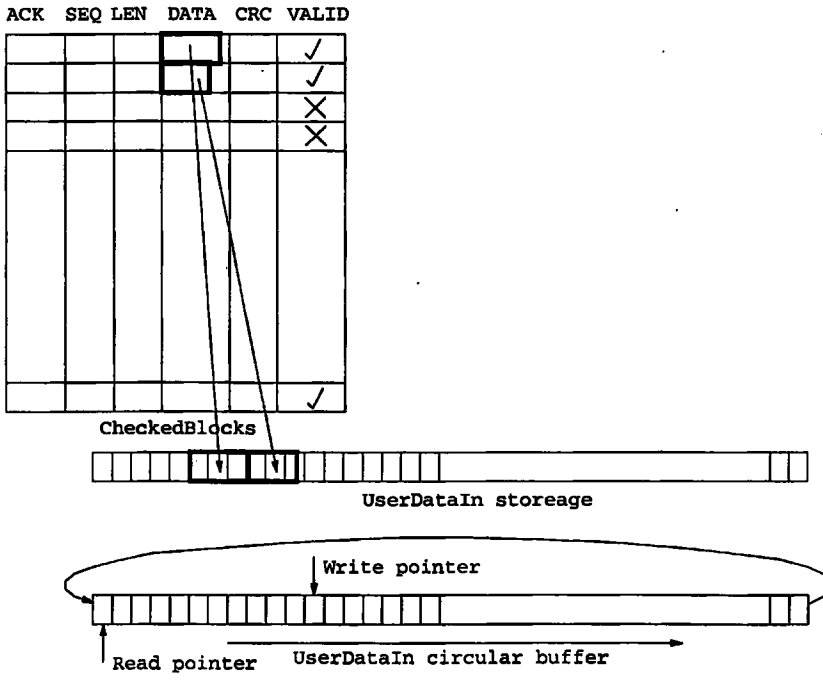


***Figure D-14* Output User Data process and *UserDataIn circular buffer*.**

Steps in Output User Data:

- If there are a group of blocks that occupy consecutive slots in *CheckedBlocks* and the first of these blocks is in slot 0 then copy LEN symbols from each of these blocks in order to *UserDataIn*.
- Once the symbols have been written shift out the associated blocks from the 'top' of *CheckedBlocks*, shuffle all other blocks 'up' and shift in new blocks at the 'bottom' with their valid flags clear.

Structure of 'Output User Data' Function.

OpUsrDat():

WHILE block in slot 0 of *CheckedBlocks* is valid (i.e. CB.b[0].valid == 1)
BEGIN(1)
Read block from slot 0 of *CheckedBlocks* and store in temporary storage.
Write DATA-field of temporary storage to *UserDataIn*.
Move all blocks in *CheckedBlocks* up 1 slot (block in slot 0 will be moved-out and slot in block '*window-size*' will be new).
Clear valid flag in block '*window-size*'.
END(1)
ENDWHILE

D.1.2.7 Synchronise Decoding.

Synchronise Decoding is responsible for preparing the receive/decode scheme to handle the next input of channel data. It keeps track of which blocks were previously transferred from *CheckedBlocks* to *UserDataIn* and can determine how many blocks have just been transferred by comparing its own internal reference with the value in *RxdBlockInfo*. Using this information it updates *DecodeLevelData*.

Steps in Synchronise Decoding:

- The difference between *RxdBlockInfo* and *OldRxdBlockInfo* indicates how many blocks *B* have been transferred from *BlocksChecked* to *UserDataIn*.
- *B* entries from *DecodeLevelData* are shuffled out (i.e. *DecodeLevelData* is shifted left *B* places) and the remaining entries are updated (1→2→3→1→1 ..... →1→1).
- Having updated the remaining entries, the *B* vacated entries in *DecodeLevelData* are set to 1.

Structure of 'Synchronise Decoding' Function.

SynchDec():

Determine how many blocks have just been written to *UserDataIn* (N = *RxdBlockInfo* - *OldRxdBlockInfo*)
Shift *DecodeLevelData* left N places - N entries are lost on the left-hand-side and N zeros moved-in on the right-hand-side with their flags set to zero.
Update each entry i in *DecodeLevelData*

Rules:

Initial Settings:	DLD.d[i]=0, DLD.f[i]=0.
To update:	If DLD.f[i] = 0 and DD.d[i] < 3 then DD.d[i] = DD.d[i] + 1
	If DD.d[i] = 3 then DD.d[i] = 1 also DD.f[i]=1.